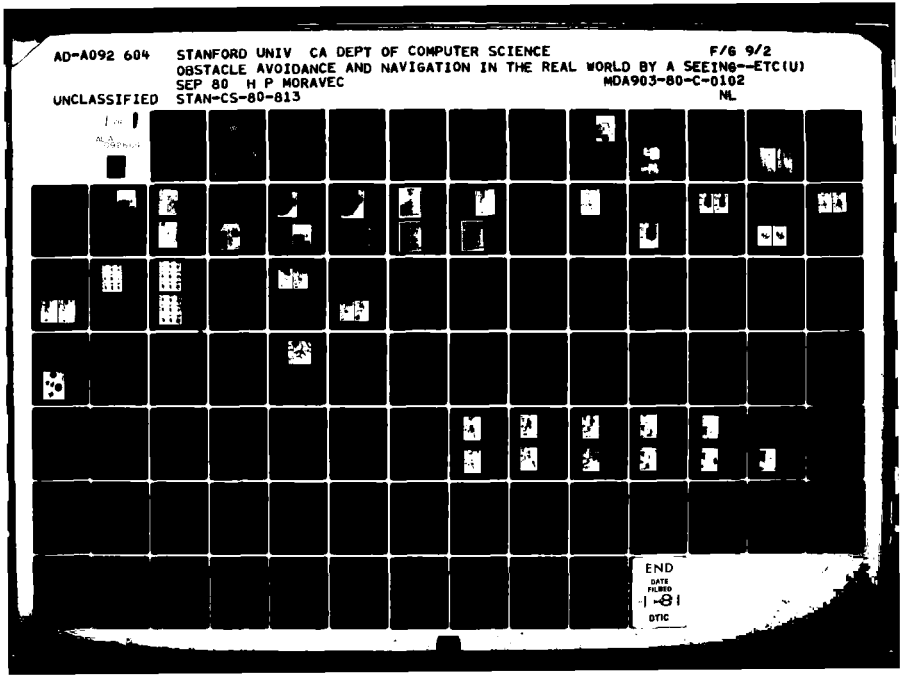
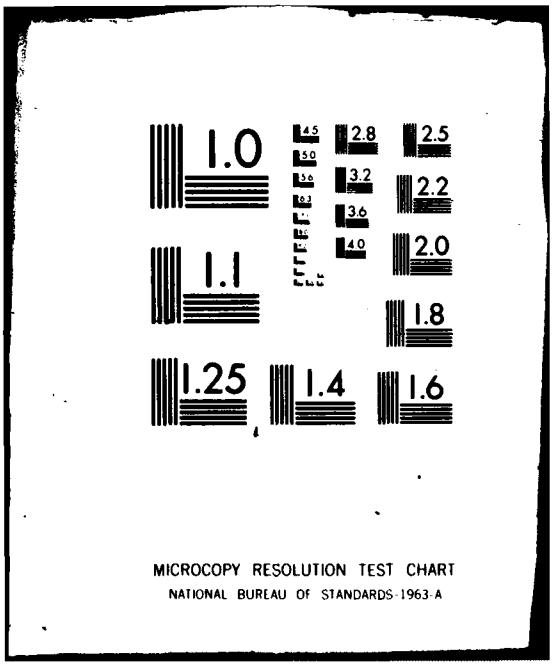

AD-A092 604
STANFORD UNIV CA DEPT OF COMPUTER SCIENCE
F/G 9/2
OBSTACLE AVOIDANCE AND NAVIGATION IN THE REAL WORLD BY A SEEING--ETC(U)
SEP 80 H P MORAVEC
MDA903-80-C-0102
UNCLASSIFIED
STAN-CS-80-813
NL
END
DATE FILMED
1-81
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Stanford Artificial Intelligence Laboratory
Memo AIM-340

Computer Science Department
Report No. STAN-CS-80-813

September 1980

Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover

by

Hans P. Moravec

Research sponsored by

Advanced Research Projects Agency
National Science Foundation
National Aeronautics and Space Administration
Jet Propulsion Laboratory

COMPUTER SCIENCE DEPARTMENT
Stanford University

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: STAN-CS-80-813 (AIM-340)

2. GOVT ACCESSION NO.: AD-A092 604

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.

5. TYPE OF REPORT & PERIOD COVERED: technical, September 1980

6. PERFORMING ORG. REPORT NUMBER: STAN-CS-80-813 (AIM-340)

7. AUTHOR(s): Hans P. Moravec

8. CONTRACT OR GRANT NUMBER(s): MDA 903-80-C-0102, NSF-DAR-78-15914

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Department of Computer Science
Stanford University
Stanford, California 94305 USA

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS:
Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Avenue, Arlington, Virginia 22209

12. REPORT DATE: Sept 1980

13. NO. OF PAGES: 174

14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office):
Mr. Philip Surra, Resident Representative
Office of Naval Research, Durand 165
Stanford University

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(see other side)

LEVEL II

AD A092604

DTIC FILE COPY

DTIC ELECTE DEC 2 1980 S C D

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. KEY WORDS (Continued)

20 ABSTRACT (Continued)

The Stanford AI Lab cart is a card-table sized mobile robot controlled remotely through a radio link, and equipped with a TV camera and transmitter. A computer has been programmed to drive the cart through cluttered indoor and outdoor spaces, gaining its knowledge of the world entirely from images broadcast by the onboard TV system.

The cart uses several kinds of stereo to locate objects around it in 3D and to deduce its own motion. It plans an obstacle avoiding path to a desired destination on the basis of a model built with this information. The plan changes as the cart perceives new obstacles on its journey.

The system is reliable for short runs, but slow. The cart moves one meter every ten to fifteen minutes, in lurches. After rolling a meter it stops, takes some pictures and thinks about them for a long time. Then it plans a new path, executes a little of it, and pauses again.

The program has successfully driven the cart through several 20 meter indoor courses (each taking about five hours) complex enough to necessitate three or four avoiding swerves. A less successful outdoor run, in which the cart skirted two obstacles but collided with a third, was also done. Harsh lighting (very bright surfaces next to very dark shadows) giving poor pictures and movement of shadows during the cart's creeping progress were major reasons for the poorer outdoor performance. The *action* portions of these runs were filmed by computer controlled cameras.

Stanford Artificial Intelligence Laboratory
Memo AIM-340

Computer Science Department
Report No. STAN-CS-80-813

September 1980

Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover

Hans P. Moravec

ABSTRACT

The Stanford AI Lab cart is a card-table sized mobile robot controlled remotely through a radio link, and equipped with a TV camera and transmitter. A computer has been programmed to drive the cart through cluttered indoor and outdoor spaces, gaining its knowledge of the world entirely from images broadcast by the onboard TV system.

The cart uses several kinds of stereo to locate objects around it in 3D and to deduce its own motion. It plans an obstacle avoiding path to a desired destination on the basis of a model built with this information. The plan changes as the cart perceives new obstacles on its journey.

The system is reliable for short runs, but slow. The cart moves one meter every ten to fifteen minutes, in lurches. After rolling a meter it stops, takes some pictures and thinks about them for a long time. Then it plans a new path, executes a little of it, and pauses again.

The program has successfully driven the cart through several 20 meter indoor courses (each taking about five hours) complex enough to necessitate three or four avoiding swerves. A less successful outdoor run, in which the cart skirted two obstacles but collided with a third, was also done. Harsh lighting (very bright surfaces next to very dark shadows) giving poor pictures and movement of shadows during the cart's creeping progress were major reasons for the poorer outdoor performance. The *action* portions of these runs were filmed by computer controlled cameras.

This thesis was submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported in part by contracts and grants from the Defense Advanced Research Projects Agency, the National Science Foundation, the National Aeronautics and Space Administration and the Jet Propulsion Laboratory of the California Institute of Technology.

The views and conclusions contained in this paper are those of the author, and in no way reflect the official policies, either expressed or implied, of Stanford University, any agency of the U.S. government or any other body whatsoever.

Acknowledgements

My nine year stay at the Stanford AI lab has been pleasant, but long enough to tax my memory. I hope not too many people have been forgotten.

Rod Brooks helped with most aspects of this work during the last two years and especially during the grueling final weeks before the lab move in 1979. Without his help my PhD-hood might have taken ten years.

Vic Scheinman has been a patron saint of the cart project since well before my involvement. Over the years he has provided untold many motor and sensor assemblies, and general mechanical expertise whenever requested. His latest contribution was the camera slider assembly which is the backbone of the cart's vision.

Mike Farmwald wrote several key routines in the display and vision software packages used by the obstacle avoider, and helped construct some of the physical environment which made cart operations pleasant.

Jeff Rubin pleasantly helped with the electronic design of the radio control link and other major components.

Marvin Horton provided support and an array of camera equiment, including an impressive home built ten meter hydraulic movie crane for the filming of the final cart runs.

Others who have helped recently are Harlyn Baker, Peter Blicher, Dick Gabriel, Bill Gosper, Elaine Kant, Mark LeBrun, Robert Maas, Allan Miller, Lynne Toribara and Polle Zellweger.

My debts in the farther past are many, and my recollection is sporadic. I remember particularly the difficult time reconstructing the cart's TV transmitter. Bruce Bullock, Tom Gafford, Ed McGuire and Lynn Quam made it somewhat less traumatic.

Delving even farther, I wish to thank Bruce Baumgart for radiating a pleasantly (and constructively) wild eyed attitude about this line of work, and Rod Schmidt, whom I have never met, for building the hardware that made my first five years of cart work possible.

In addition I owe very much to the unrestrictive atmosphere created at the lab mainly by John McCarthy and Les Earnest, and maintained by Tom Binford, and also to the excellent system support provided to me (over the years) by Marty Frost, Ralph Gorin, Ted Panofsky and Robert Poor.

Table of Contents

Introduction

This is a report about a modest attempt at endowing a mild mannered machine with a few of the attributes of higher animals.

An electric vehicle, called the cart, remote controlled by a computer, and equipped with a TV camera through which the computer can see, has been programmed to run undemanding but realistic obstacle courses.

The methods used are minimal and crude, and the design criteria were simplicity and performance. The work is seen as an evolutionary step on the road to intellectual development in machines. Similar humble experiments in early vertebrates eventually resulted in human beings.

The hardware is also minimal. The television camera is the cart's only sense organ. The picture perceived can be converted to an array of numbers in the computer of about 256 rows and 256 columns, with each number representing up to 64 shades of gray. The cart can drive forwards and back, steer its front wheels and move its camera from side to side. The computer controls these functions by turning motors on and off for specific lengths of time.

Better (at least more expensive) hardware has been and is being used in similar work elsewhere. SRI's Shakey moved around in a contrived world of giant

blocks and clean walls. JPL is trying to develop a semi-autonomous rover for the exploration of Mars and other far away places (the project is currently mothballed awaiting resumption of funding). Both SRI's and JPL's robots use laser rangefinders to determine the distance of nearby objects in a fairly direct manner. My system, using less hardware and more computation, extracts the distance information from a series of still pictures of the world from different points of view, by noting the relative displacement of objects from one picture to the next.

Applications

A Mars rover is the most likely near term use for robot vehicle techniques.

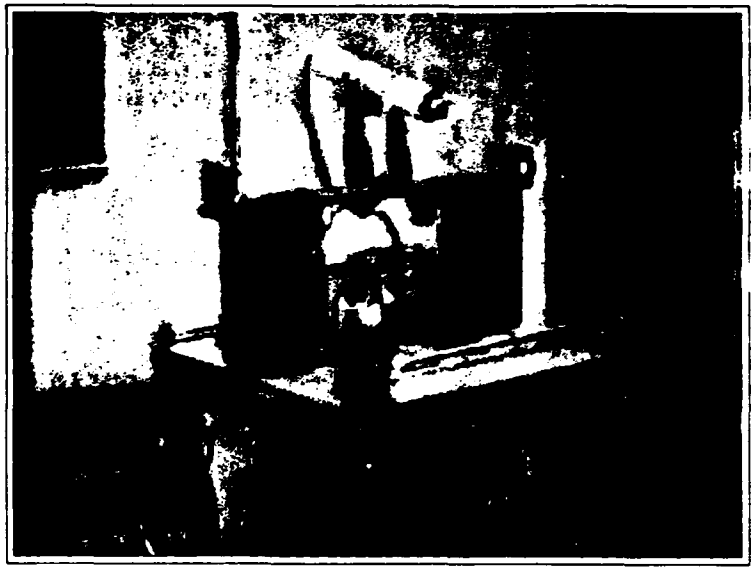


Figure 1-1: *The cart, like a card table, but taller*

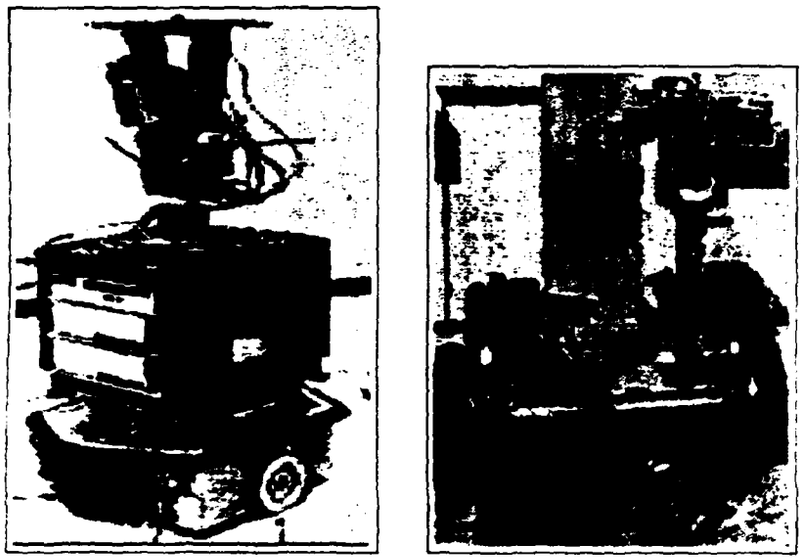


Figure 1-2: *SRI's Shakey and JPL's Robotics Research Vehicle*

The half hour radio delay between Earth and Mars makes direct remote control an unsatisfactory way of guiding an exploring device. Automatic aids, however limited, would greatly extend its capabilities. I see my methods as complementary to approaches based on rangefinders. A robot explorer will have a camera in addition to whatever other sensors it carries. Visual obstacle avoidance can be used to enhance the reliability of other methods, and to provide a backup for them.

Robot submersibles are almost as exotic as Mars rovers, and may represent another not so distant application of related methods. Remote control of submersibles is difficult because water attenuates conventional forms of long distance

communication. Semi-autonomous minisubs could be useful for some kinds of exploration and may finally make seabed mining practical.

In the longer run the fruits of this kind of work can be expected to find less exotic uses. Range finder approaches to locating obstacles are simpler because they directly provide the small amount of information needed for undemanding tasks. As the quantity of information to be extracted increases the amount of processing, regardless of the exact nature of the sensor, will also increase.

What a smart robot thinks about the world shouldn't be affected too much by exactly what it sees with. Low level processing differences will be mostly gone at intermediate and high levels. Present cameras offer a more detailed description of the world than contemporary rangefinders and camera based techniques probably have more potential for higher visual functions.

The mundane applications are more demanding than the rover task. A machine that navigates in the crowded everyday world, whether a robot servant or an automatic car, must efficiently recognize many of the things it encounters to be safe and effective. This will require methods and processing power beyond those now existing. The additional need for low cost guarantees they will be a while in coming. On the other hand work similar to mine will eventually make them feasible.

History

This work was shaped to a great extent by its physical circumstances; the nature and limitations of the cart vehicle itself, and the resources that could be brought to bear on it. The cart has always been the poor relation of the Stanford Hand-Eye project, and has suffered from lack of many things, not the least of which was sufficient commitment and respect by any principal investigator.

The cart was built in the early 1960's by a group in the Stanford Mechanical Engineering Department under a NASA contract, to investigate potential solutions for the problems of remote controlling a lunar rover from Earth. The image from an onboard TV camera was broadcast to a human operator who manipulated a steering control. The control signals were delayed for two and a half seconds by a tape loop, then broadcast to the cart, simulating the Earth/Moon round trip delay.

The AI lab, then in its enthusiastic spit and baling wire infancy, acquired the cart gratis from ME after they were done, minus its video and remote control electronics. Rod Schmidt, an EE graduate student and radio amateur was induced to work on restoring the vehicle, and driving it under computer control. He spent over two years, but little money, single-handedly building a radio control link based on a model airplane controller, and a UHF TV link. The control

link was relatively straightforward, the video receiver was a modified TV set, but the UHF TV transmitter took 18 laborious months of tweaking tiny capacitors and half centimeter turns of wire. The resulting robot was ugly, reflecting its rushed assembly, and marginally functional (the airplane proportional controller was very inaccurate). Like an old car, it needed (and needs) constant repair and replacement of parts, major and minor, that break.

Schmidt then wrote a program for the PDP-6 which drove the cart in real time (but with its motors set to run very slowly) along a wide white line. It worked occasionally. Following a white line with a raised TV camera and a computer turns out to be much more difficult than following a line at close range with a photocell tracker. The camera scene is full of high contrast extraneous detail, and the lighting conditions are unreliable. This simple program taxed the processing power of the PDP-6. It also clearly demonstrated the need for more accurate and reliable hardware if more ambitious navigation problems were to be tackled. Schmidt wrote up the results and finished his degree.

Bruce Baumgart picked up the cart banner, and announced an ambitious approach that would involve modelling the world in great detail, and by which the cart could deduce its position by comparing the image it saw through its camera with images produced from its model by a 3D drawing program. He succeeded reasonably well with the graphics end of the problem.

The real world part was a dismal failure. He began with a rebuild of the cart control electronics, replacing the very inaccurate analog link with a supposedly more repeatable digital one. He worked as single-handedly as did Schmidt, but without the benefit of prior experience with hardware construction. The end result was a control link that, because of a combination of design flaws and un-

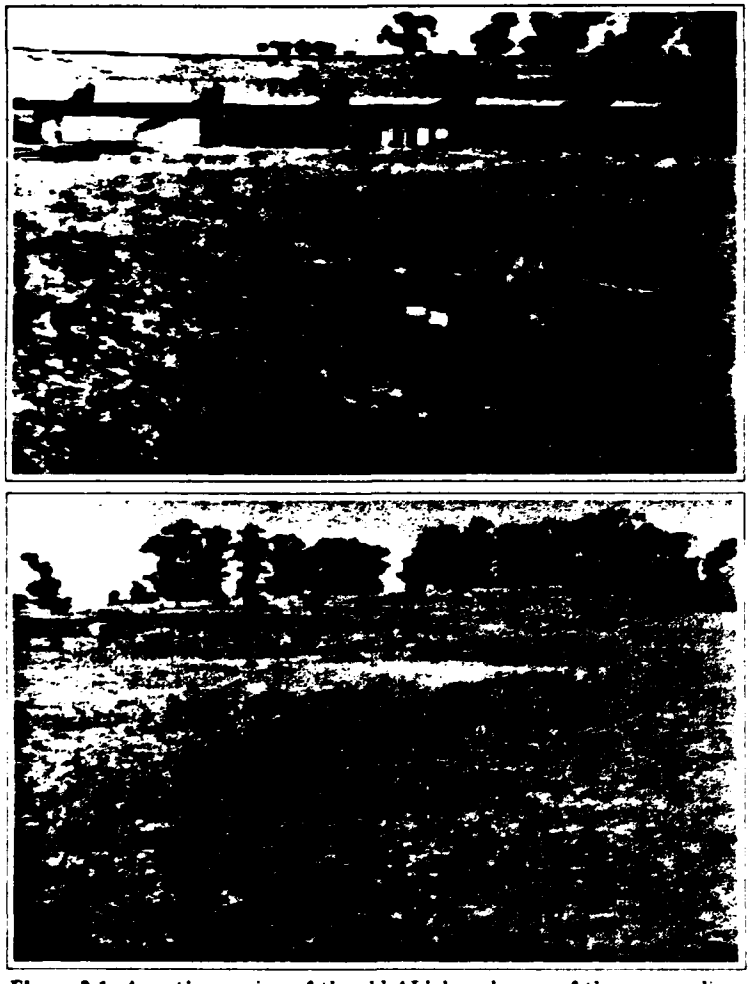


Figure 2-1: *A cart's eye view of the old AI lab and some of the surrounding terrain, digitised during the Baumgart era.*

detected bugs, was virtually unusable. One time out of three the cart moved in a direction opposite to which it had been commanded, left for right or forwards for backwards.

During this period a number of incoming students were assigned to the "cart project". Each correctly perceived the situation within a year, and went on to something else. The cart's reputation as a serious piece of research apparatus, never too high, sank to new depths.

I came to the AI lab, enthusiastic and naïve, with the specific intention of working with the cart. I'd built a series of small robots, beginning in elementary school, and the cart, of whose existence, but not exact condition, I'd learned, seemed like the logical next step. Conditions at the lab were liberal enough that my choice was not met with termination of financial support, but this largesse did not easily extend to equipment purchases.

Lynn Quam, who had done considerable work with stereo mapping from pictures from the Mariners 6 and 7 Mars missions, expressed an interest in the cart around this time, for its clear research value for Mars rovers. We agreed to split up the problem (the exact goals for the collaboration were never completely clear; mainly they were to get the cart to do as much as possible). He would do the vision, and I would get the control hardware working adequately and write motor subroutines which could translate commands like move a meter forward and a half to the right into appropriate steering and drive signals.

I debugged, then re-designed and rebuilt the control link to work reliably, and wrote a routine that incorporated a simulation of the cart, to drive it (this subroutine was resurrected in the final months of the obstacle avoider effort, and is described in chapter 8). I was very elated by my quick success, and spent con-

siderable time taking the cart on joy rides. I would open the big machine room doors near the cart's parking place, and turn on the cart. Then I would rush to my office, tune in the cart video signal on a monitor, start a remote control program, and, in armchair and air conditioned comfort, drive the cart out the doors. I would steer it along the outer deck of the lab to one of three ramps on different sides of the building. I then drove it down the narrow ramp (they were built for deliveries), and out into the driveway or onto the grass, to see (on my screen) what there was to see. Later I would drive it back the same way, then get up to close the doors and power it down. With increasing experience, I became increasingly cocky. During the 1973 IJCAI, held at Stanford, I repeatedly drove it up and down the ramps, and elsewhere, for the amusement of the crowds visiting the AI lab during an IJCAI sponsored winetasting.

Shortly after the IJCAI my luck ran out. Preparing to drive it down the front ramp for a demonstration, I misjudged the position of the right edge by a few centimeters. The cart's right wheels missed the ramp, and the picture on my screen slowly rotated 90°, then turned into noise. Outside, the cart was lying on its side, with acid from its batteries spilling into the electronics. Sigh.

The sealed TV camera was not damaged. The control link took less than a month to resurrect. Schmidt's video transmitter was another matter. I spent a total of nine frustrating months first trying, unsuccessfully, to repair it, then building (and repeatedly rebuilding) a new one from the old parts using a cleaner design found in a ham magazine and a newly announced UHF amplifier module from RCA. The new one almost worked, though its tuning was touchy. The major problem was a distortion in the modulation. The RCA module was designed for FM, and did a poor job on the AM video signal. Although TV sets found the broadcast tolerable, our video digitizer was too finicky

During these nine difficult months I wrote to potential manufacturers of such transmitters, and also inquired about borrowing the video link used by Shakey, which had been retired by SRI. SRI, after due deliberation, turned me down. Small video transmitters are not off the shelf items; the best commercial offer I got was for a two watt transmitter costing $4000.

Four kilobucks was an order of magnitude more money than had ever been put into cart hardware by the AI lab, though it was was considerably less than had been spent on salary in Schmidt's 18 months and my 9 months of transmitter hacking. I begged for it and got an agreement from John McCarthy that I could buy a transmitter, using ARPA money, after demonstrating a capability to do vision.

During the next month I wrote a program that picked a number of features in one picture (the "interest operator" of Chapter 5 was invented here) of a motion stereo pair, and found them in the other image with a simple correlator, did a crude distance calculation, and generated a fancy display. Apparently this was satisfactory; the transmitter was ordered.

By this time Quam had gone on to other things. With the cart once again functional, I wrote a program that drove it down the road in a straight line by servoing on points it found on the distant horizon with the interest operator and tracked with the correlator. Like the current obstacle avoider, it did not run in real time, but in lurches. That task was much easier, and even on the KA-10, our main processor at the time, each lurch took at most 15 seconds of real time. The distance travelled per lurch was variable; as small as a quarter meter when the program detected significant variations from its desired straight path, repeatedly doubling up to many meters when everything seemed to be working. The program

also observed the cart's response to commands, and updated a response model which it used to guide future commands. The program was reliable and fun to watch, except that the remote control link occasionally failed badly. The cause appeared to be interference from passing CBers. The citizens band boom had started, and our 100 milliwatt control link, which operated in the CB band, was not up to the competition.

I replaced the model airplane transmitter and receiver by standard (but modified) CB transceivers, increasing the broadcast power to 5 watts. To test this and a few other improvements in the hardware, I wrote an updated version of the horizon tracker which incorporated a new idea, the faster and more powerful "binary search" correlator of Chapter 6. This was successful, and I was ready for bigger game.

Obstacle avoidance could be accomplished using many of the techniques in the horizon tracker. A dense cloud of features on objects in the world could be tracked as the cart rolled forward, and a 3D model of their position and the cart's motion through them could be deduced from their relative motion in the image. Don Gennery had already written a camera solving subroutine, used by Quam and Hannah, which was capable of such a calculation.

I wrote a program which drove the cart, tracking features near and far, and feeding them to Gennery's subroutine. The results were disappointing. Even after substantial effort, aggravated by having only a very poor a priori model of cart motion, enough of the inevitable correlation errors escaped detection to make the camera solver converge to the wrong answer about 10 to 20% of the time. This error rate was too high for a vehicle that would need to navigate through at least tens of such steps. Around this time I happened to catch some small lizards,

that I kept for a while in a terrarium. Watching them, I observed an interesting behavior.

The lizards caught flies by pouncing on them. Since flies are fast, this requires speed and 3D precision. Each lizard had eyes on opposite sides of its head; the visual fields could not overlap significantly, ruling out stereo vision. But before a pounce, a lizard would fix an eye on its victim, and sway its head slowly from side to side. This seemed a sensible way to range.

My obstacle avoiding task was defeating the motion stereo approach, and the lizard's solution seemed promising. I built a stepping motor mechanism that could slide the cart's camera from side to side in precise increments. The highly redundant information available from this apparatus broke the back of the problem, and made the obstacle avoider that is the subject of this thesis possible.

CHAPTER 3

Overview

A typical run of the avoider system begins with a calibration of the cart's camera. The cart is parked in a standard position in front of a wall of spots. A calibration program (described in Chapter 4) notes the disparity in position of the spots in the image seen by the camera with their position predicted from an idealized model of the situation. It calculates a distortion correction polynomial which relates these positions, and which is used in subsequent ranging calculations.

The cart is then manually driven to its obstacle course. Typically this is either in the large room in which it lives, or a stretch of the driveway which encircles the AI lab. Chairs, boxes, cardboard constructions and assorted debris serve as obstacles in the room. Outdoors the course contains curbing, trees, parked cars and signposts as well.

The obstacle avoiding program is started. It begins by asking for the cart's destination, relative to its current position and heading. After being told, say, 50 meters forward and 20 to the right, it begins its maneuvers.

It activates a mechanism which moves the TV camera, and digitizes about nine pictures as the camera slides (in precise steps) from one side to the other along a 50 cm track.

A subroutine called the *interest operator* (described in Chapter 5) is applied to the one of these pictures. It picks out 30 or so particularly distinctive regions (features) in this picture. Another routine called the *correlator* (Chapter 6) looks for these same regions in the other frames. A program called the camera *solver* (Chapter 7) determines the three dimensional position of the features with respect to the cart from their apparent movement image to image.

The *navigator* (Chapter 8) plans a path to the destination which avoids all the perceived features by a large safety margin. The program then sends steering and drive commands to the cart to move it about a meter along the planned path. The cart's response to such commands is not very precise.

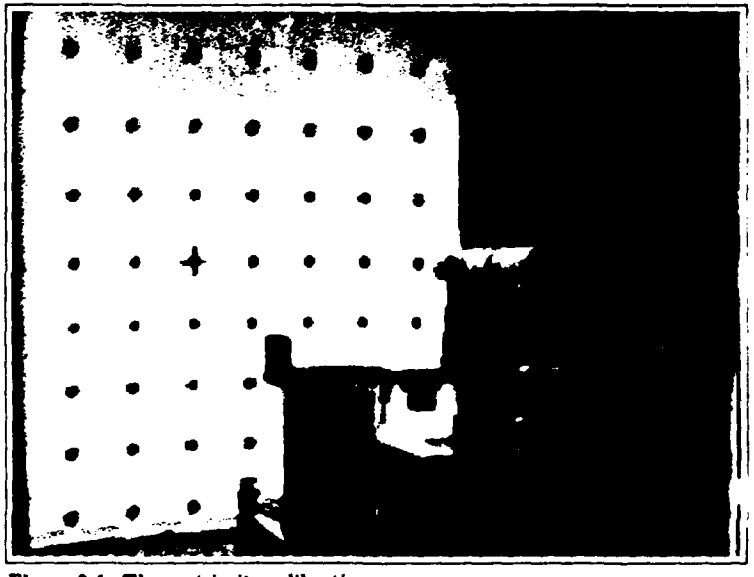


Figure 3-1: *The cart in its calibration pose*

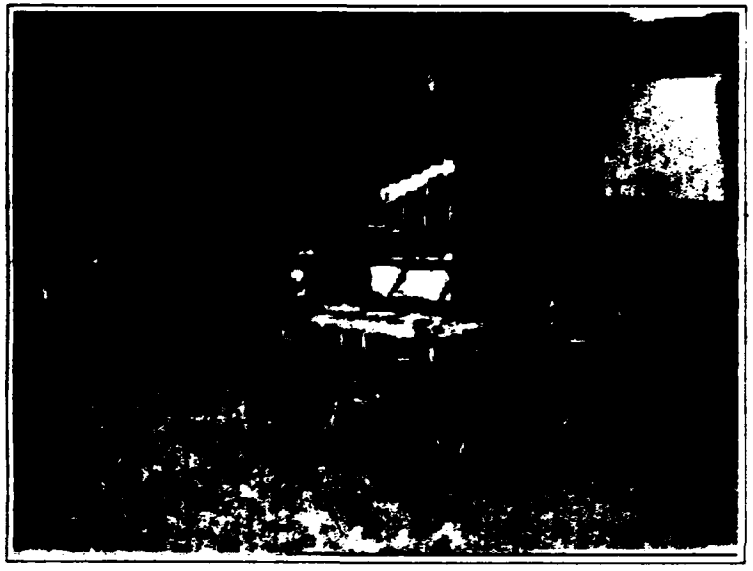


Figure 3-2: *The cart indoors*

After the step forward the camera is operated as before, and nine new images are acquired. The control program uses a version of the correlator to find as many of the features from the previous location as possible in the new pictures, and applies the camera solver. The program then deduces the cart's actual motion during the step from the apparent three dimensional shift of these features.

The motion of the cart as a whole is larger and less constrained than the precise slide of the camera. The images between steps forward can vary greatly, and the correlator is usually unable to find many of the features it wants. The interest operator/correlator/camera solver combination is used to find new features to replace lost ones.

Figure 3-3: *The cart outdoors*

The three dimensional location of any new features found is added to the program's model of the world. The navigator is invoked to generate a new path that avoids all known features, and the cart is commanded to take another step forward.

This continues until the cart arrives at its destination or until some disaster terminates the program.

Appendix 3 documents the evolution of the cart's internal world model in response to the scenery during a sample run.

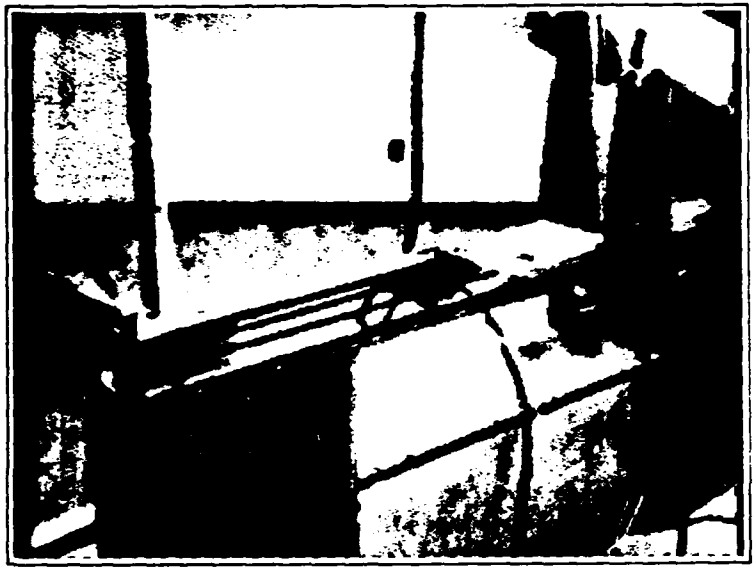


Figure 3-4: *A closeup of the translate mechanism*

An Objection

A method as simple as this is unlikely to handle every situation well. The most obvious problem is the apparently random choice of features tracked. If the interest operator happens to avoid choosing any points on a given obstruction, the program will never notice it, and might plan a path right through it.

The interest operator was designed to minimise this danger. It chooses a relatively uniform scattering of points over the image, locally picking those with most contrast. Effectively it samples the picture at low resolution, indicating the most promising regions in each sample area.

Objects lying in the path of the vehicle occupy ever larger areas of the camera image as the cart rolls forward. The interest operator is applied repeatedly, and the probability that it will choose a feature or two on the obstacle increases correspondingly. Typical obstructions are generally detected before its too late. Very small or very smooth objects are sometimes overlooked.

Calibration

The cart camera, like most vidicons, has peculiar geometric properties. Its precision has been enhanced by an automatic focal length and distortion deter-

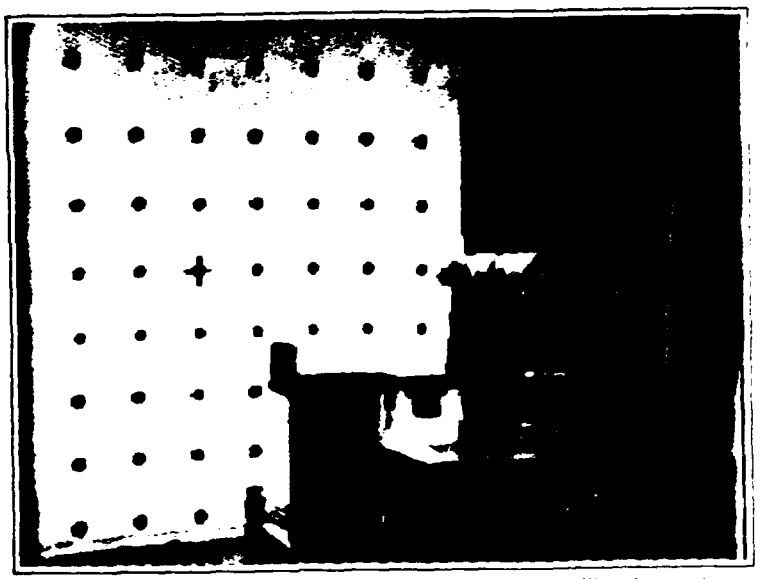


Figure 4-1: *The cart in its calibration posture before the calibration pattern. A program automatically locates the cross and the spots, and deduces the camera's focal length and distortion.*

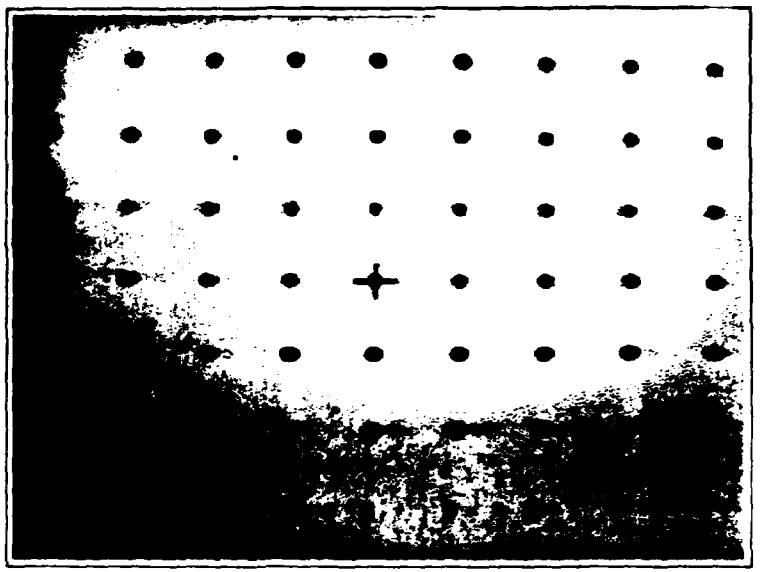


Figure 4-2: *The spot array, as digitised by the cart camera.*

mining program.

The cart is parked a precise distance in front of a wall of many spots and one cross (Figure 4-1). The program digitises an image of the spot array, locates the spots and the cross, and constructs a a two dimensional polynomial that relates the position of the spots in the image to their position in an ideal unity focal length camera, and another polynomial that converts points from the ideal camera to points in the image. These polynomials are used to correct the positions of perceived objects in later scenes.

The program tolerates a wide range of spot parameters (about 3 to 12 spots

across), arbitrary image rotation, and is very robust. After being intensely fiddled with to work successfully on an initial set of 20 widely varying images, it has worked without error on 50 successive images. The test pattern for the cart is a 3 meter square painted on a wall, with 5 cm spots at 30 cm intervals. The program has also been used successfully with a small array (22 x 28 cm) to calibrate cameras other than the cart's [W1].

The algorithm reads in an image of such an array, and begins by determining its approximate spacing and orientation. It trims the picture to make it square, reduces it by averaging to 64 by 64, calculates the Fourier transform of the reduced image and takes its power spectrum, arriving at a 2D transform symmetric about the origin, and having strong peaks at frequencies corresponding to the horisontal and vertical and half-diagonal spacings, with weaker peaks at the harmonics. It multiplies each point $[i, j]$ in this transform by point $[-j, i]$ and points $[j - i, j + i]$ and $[i + j, j - i]$, effectively folding the primary peaks

Figure 4-3: *Power spectrum of Figure 4-2, and folded transform*

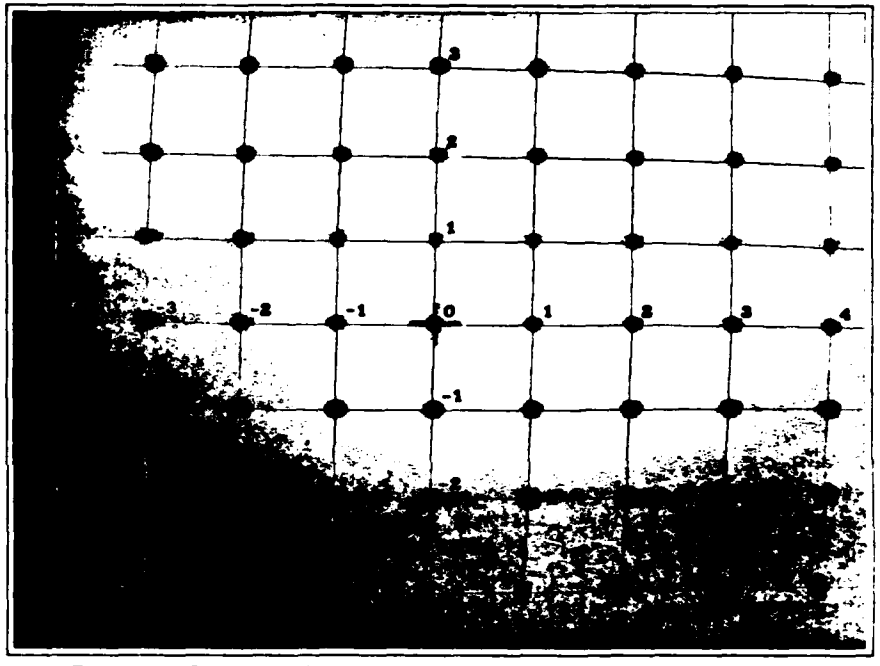


Figure 4-4: *Results of the calibration program. The distortion polynomial it produced has been used to map an undistorted grid of ideal spot positions into the calculated real world ones. The result is superimposed on the original digitised spot image, making any discrepancies obvious.*

onto one another. The strongest peak in the 90° wedge around the y axis gives the spacing and orientation information needed by the next part of the process.

The directional variance interest operator described later (Chapter 5) is applied to roughly locate a spot near the center of the image. A special operator examines a window surrounding this position, generates a histogram of intensity

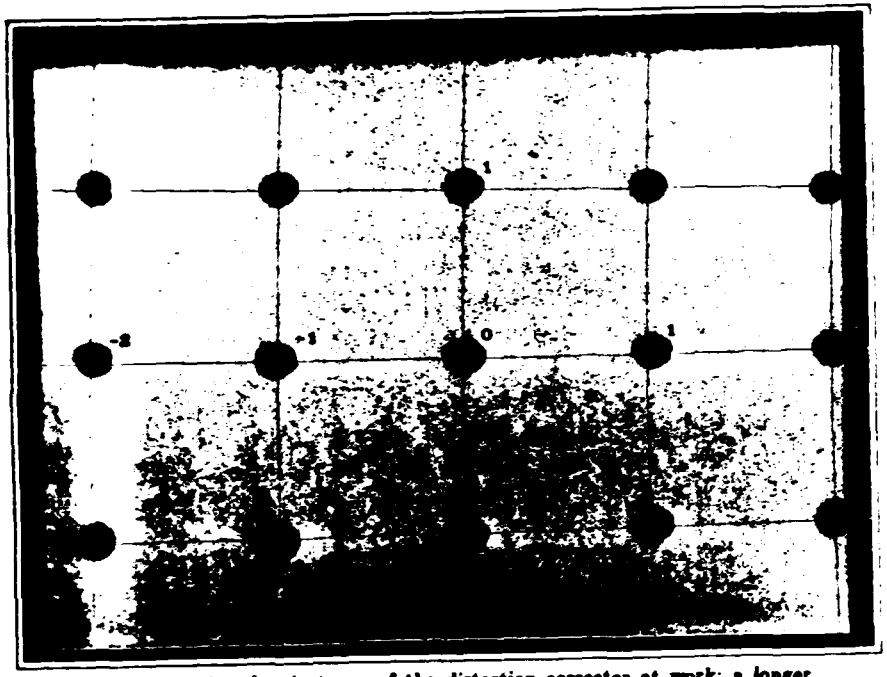


Figure 4-5: *Another instance of the distortion corrector at work; a longer focal length lens*

values within the window, decides a threshold for separating the black spot from the white background, and calculates the centroid and first and second moment of the spot. This operator is again applied at a displacement from the first centroid indicated by the orientation and spacing of the grid, and so on, the region of found spots growing outward from the seed.

A binary template for the expected appearance of the cross in the middle of the array is constructed from the orientation/spacing determined determined by

Figure 4-6: *Yet another example; a rotation*

the Fourier transform step. The area around each of the found spots is thresholded on the basis of the expected cross area, and the resulting two valued pattern is convolved with the cross template. The closest match in the central portion of the picture is declared to be the origin.

Two least-squares polynomials (one for X and one for Y) of third (or sometimes fourth) degree in two variables, relating the actual positions of the spots to the ideal positions in a unity focal length camera, are then generated and written into a file.

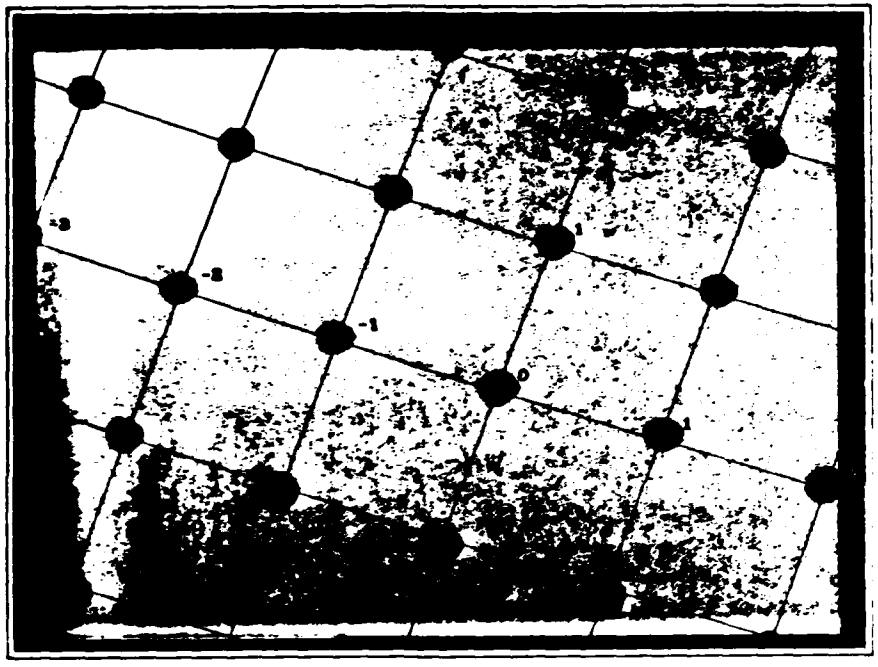


Figure 4-7: *And yet another example*

The polynomials are used in the obstacle avoider to correct for camera roll, tilt, focal length and long term variations in the vidicon geometry.

Interest Operator

The cart vision code deals with very simple primitive entities, localised regions called features. A feature is conceptually a point in the three dimensional

Figure 5-1: *A cart's eye view from the starting position of an obstacle run, and features picked out by the interest operator. They are labelled in order of decreasing interest measure.*

world, but it is found by examining localities larger than points in pictures. A feature is good if it can be located unambiguously in different views of a scene. A uniformly colored region or a simple edge does not make for good features because its parts are indistinguishable. Regions, such as corners, with high contrast in orthogonal directions are best.

New features in images are picked by a subroutine called the *interest operator*, an example of whose operation is displayed in Figure 5-1. It tries to select a relatively uniform scattering, to maximize the probability that a few features will be picked on every visible object, and to choose areas that can be easily found in other images. Both goals are achieved by returning regions that are local maxima of a directional variance measure. Featureless areas and simple edges, which have no variance in the direction of the edge, are thus avoided.

Directional variance is measured over small square windows. Sums of squares of differences of pixels adjacent in each of four directions (horizontal, vertical and two diagonals) over each window are calculated, and the window's interest measure is the minimum of these four sums.

Features are chosen where the interest measure has local maxima. The feature is conceptually the point at the center of the window with this locally maximal value.

This measure is evaluated on windows spaced half a window width apart over the entire image. A window is declared to contain an interesting feature if its variance measure is a local maximum, that is, if it has the largest value of the twenty five windows which overlap or contact it.

The variance measure depends on adjacent pixel differences and responds

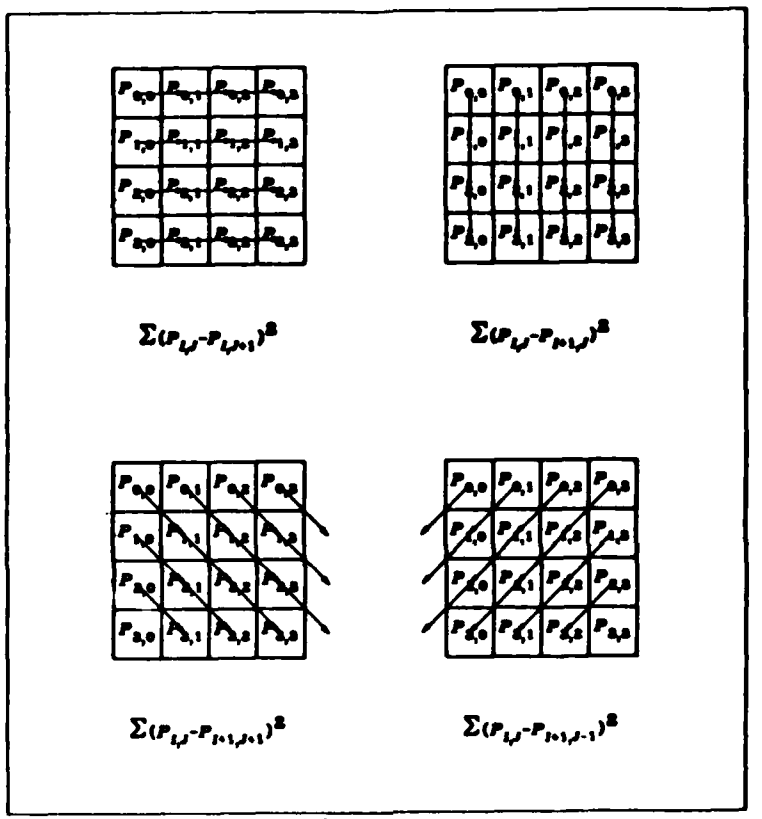


Figure 5-2: *A typical interest operator window, and the four sums calculated over it ($P_{I,J}$ are the pixel brightnesses). The interest measure of the window is the minimum of the four sums.*

to high frequency noise in the image. The effects of noise are alleviated and the processing time is shortened by applying the operator to a reduced image. In the current program original images are 240 lines high by 256 pixels wide. The

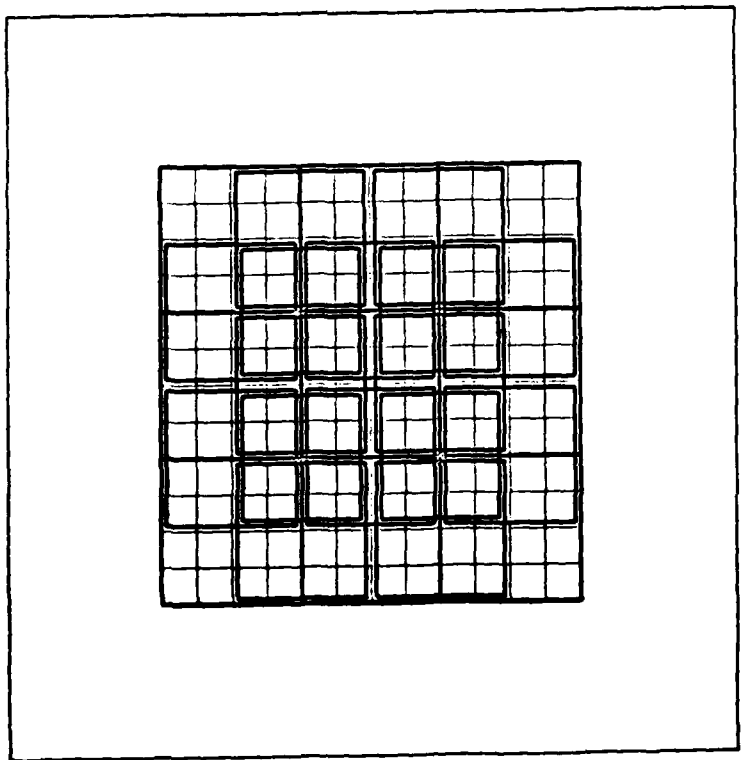


Figure 5-3: *The twenty five overlapping windows considered in a local maximum decision. The smallest cells in the diagram are individual pixels. The four by four array of these in the center of the image is the window being considered as a local maximum. In order for it to be chosen as a feature to track, its interest measure must equal or exceed that of each of the other outlined four by four areas.*

interest operator is applied to the 120 by 128 version, on windows 3 pixels square.

The local maxima found are stored in an array, sorted in order of decreasing

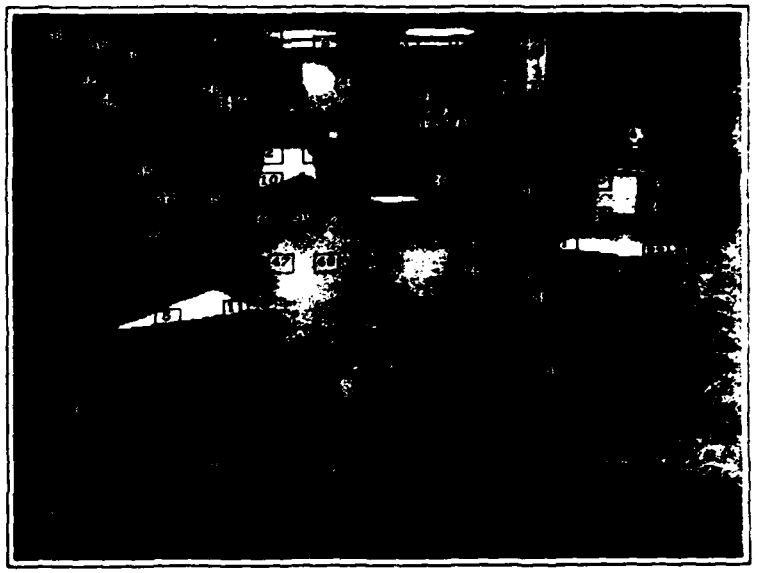


Figure 5-4: *Another obstacle run interest operator application*

variance.

The entire process on a typical 260 by 240 image, using 6 by 6 windows takes about 75 milliseconds on the KL-10. The variance computation and local maximum test are coded in FAIL (our assembler) [WG1], the maxima sorting and top level are in SAIL (an Algol-like language) [R1].

Once a feature is chosen, its appearance is recorded as series of excerpts from the reduced image sequence. A window (6 by 6 in the current implementation) is excised around the feature's location from each of the variously reduced pictures. Only a tiny fraction of the area of the original (unreduced) image is extracted. Four times as much of the x2 reduced image is stored, sixteen times

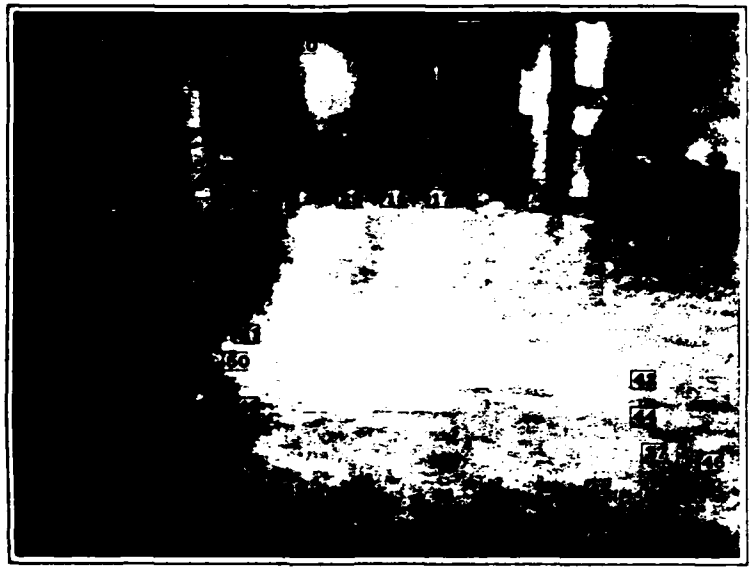


Figure 5-5: *More interest operating.*

as much of the x4 reduction, and so on until at some level we have the whole image. The final result is a series of 6 by 6 pictures, beginning with a very blurry rendition of the whole picture, gradually sooming in linear expansions of two to a sharp closeup of the feature. Of course, it records the appearance correctly from only one point of view.

Weaknesses

The interest operator has some fundamental limitations. The basic measure was chosen to reject simple edges and uniform areas. Edges are not suitable features for the correlator because the different parts of an edge are indistinguishable.

The measure is able to unambiguously reject edges only if they are oriented along the four directions of summation. Edges whose angle is an odd multiple of 22.5° give non-zero values for all four sums, and are sometimes incorrectly chosen as interesting.

The operator especially favors intersecting edges. These are sometimes corners or cracks in objects, and are very good. Sometimes they are caused by a distant object peering over the edge of a nearby one and then they are very bad. Such spurious intersections don't have a definite distance, and must be rejected during camera solving. In general they reduce the reliability of the system.

Desirable Improvements

The operator has a fundamental and central role in the obstacle avoider, and is worth improving. Edge rejection at odd angles should be increased, maybe by generating sums in the 22.5° directions.

Rejecting near/far object intersections more reliably than the current implementation does is possible. An operator that recognized that the variance in a window was restricted to one side of an edge in that window would be a good start. Really good solutions to this problem are probably computationally much more expensive than my measure.

CHAPTER 6

Correlation

Deducing the 3D location of features from their projections in 2D images requires that we know their position in two or more such images.

The correlator is a subroutine that, given a description of a feature as produced by the interest operator from one image, finds the best match in a different, but similar, image. Its search area can be the entire new picture, or a rectangular sub-window.

The search uses a coarse to fine strategy, illustrated in Figure 6-1, that begins in reduced versions of the pictures. Typically the first step takes place at the x16 (linear) reduction level. The 6 by 6 window at that level in the feature description, that covers about one seventh of the total area of the original picture, is convolved with the search area in the correspondingly reduced version of the second picture. The 6 by 6 description patch is moved pixel by pixel over the approximately 15 by 16 destination picture, and a correlation coefficient is calculated for each trial position.

The position with the best match is recorded. The 6x6 area it occupies in the second picture is mapped to the x8 reduction level, where the corresponding region is 12 pixels by 12. The 6 by 6 window in the x8 reduced level of the feature description is then convolved with this 12 by 12 area, and the position of best

Figure 6-1: *Areas matched in a binary search correlation. Picture at top contains originally chosen feature. The outlined areas in it are the prototypes which are searched for in the bottom picture. The largest rectangle is matched first, and the area of best match in the second picture becomes the search area for the next smaller rectangle. The larger the rectangle, the lower the resolution of the pictures in which the matching is done.*

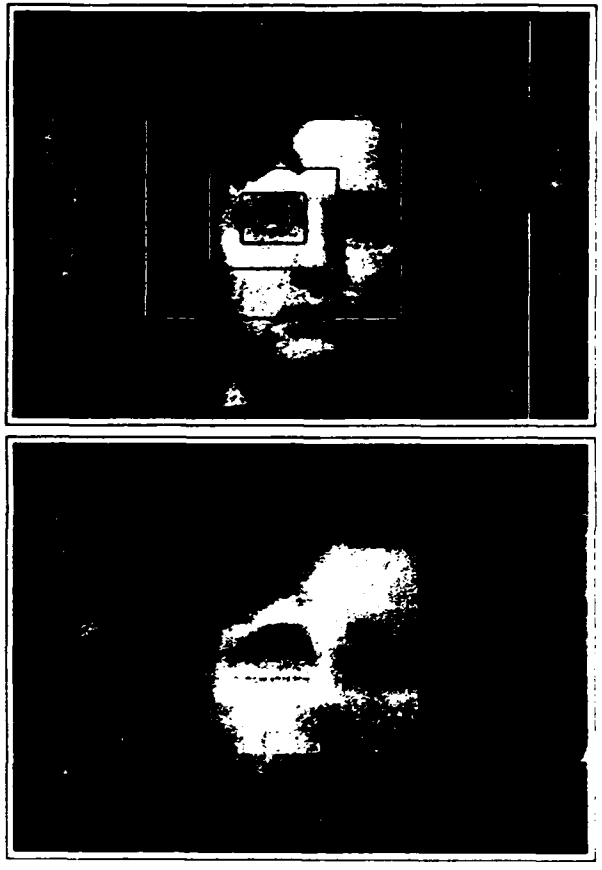


Figure 6-2: *The "conventional" representation of a feature (above) used in documents such as this one, and a more realistic version which graphically demonstrates the reduced resolution of the larger windows. The bottom picture was reconstructed entirely from the window sequence used with a binary search correlation. The coarse outer windows were interpolated to reduce quantisation artifacts.*

match is recorded and used as a search area for the x4 level.

The process continues, matching smaller and smaller, but more and more detailed windows until a 6 by 6 area is selected in the unreduced picture.

The work at each level is about the same, finding a 6 by 6 window in a 12 by 12 search area. It involves 49 summations of 36 quantities. In our example there were 5 such levels. The correlation measure used is $2\sum ab/(\sum a^2 + \sum b^2)$, where a and b are the values of pixels in the two windows being compared, with the mean of windows subtracted out, and the sums are taken over the 36 elements of a 6 by 6 window. The measure has limited tolerance to contrast differences.

The window sizes and other parameters are sometimes different from the ones used in this example.

In general, the program thus locates a huge general area around the feature in a very coarse version of the images, and successively refines the position, finding smaller and smaller areas in finer and finer representations. For windows of size n, the work at each level is approximately that of finding an n by n window in a $2n$ by $2n$ area, and there are $\log_2(w/n)$ levels, where w is the smaller dimension of the search rectangle, in unreduced picture pixels.

This approach has many advantages over a simple pass of of a correlation coefficient computation over the search window. The most obvious is speed. A scan of an 8 by 8 window over a 256 by 256 picture would require 249x249x8x8 comparisons of individual pixels. The binary method needs only about 5x81x8x8, about 150 times fewer. The advantage is lower for smaller search areas. Perhaps more important is the fact that the simple method exhibits a serious jigsaw puzzle effect. The 8 by 8 patch is matched without any reference to context, and a

match is often found in totally unrelated parts of the picture. The binary search technique uses the general context to guide the high resolution comparisons. This makes possible yet another *speedup*, because smaller windows can be used. Window sizes as small as 2 by 2 work reasonably well. The searches at very coarse levels rarely return mismatches, possibly because noise is averaged out in the reduction process, causing comparisons to be more stable. Reduced images are *also more* tolerant of geometric distortions.

The current routine uses a measure for the measure for the cross correlation which I call *pseudo normalised*, given by the formula

$$\frac{2\sum ab}{\sum a^2 + \sum b^2}$$

that has limited contrast sensitivity, avoids the degeneracies of normalised correlation on informationless windows, and is slightly cheaper to compute. A description of its derivation may be found in Appendix 6.

Timing

The formula above is expressed in terms of A and B with the means subtracted out. It can be translated into an expression involving $\sum A$, $\sum A^2$, $\sum B$, $\sum B^2$ and $\sum (A-B)^2$. By evaluating the terms involving only A, the source window, outside of the main correlation loop, the work in the inner loop can be reduced to evaluating $\sum B$, $\sum B^2$ and $\sum (A-B)^2$. *This is done in three* PDP-10 machine instructions per point by using a table in which entry i contains both i and i^2 in subfields, and by generating in-line code representing the source window, three instructions per pixel, eliminating the need for inner loop end tests and enabling the $A-B$ *computation to be done during indexing.*

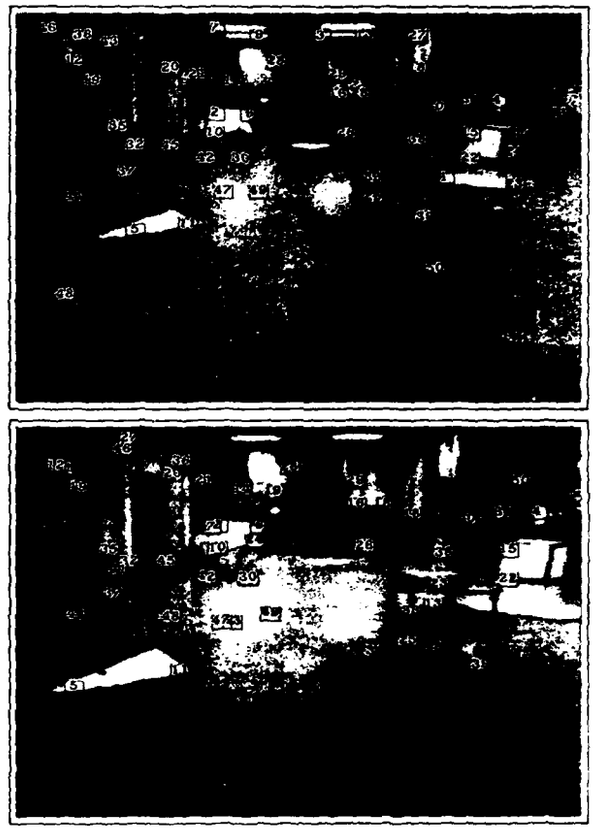


Figure 6-3: *Example of the correlator's performance on a difficult example. The interest operator has chosen features in the upper image, and the correlator has attempted to find corresponding regions in the lower one. The cart moved about one and a half meters forward between the images. Some mistakes are evident. The correlator had no a-priori knowledge about the relationship of the two images and the entire second image was searched for each feature.*

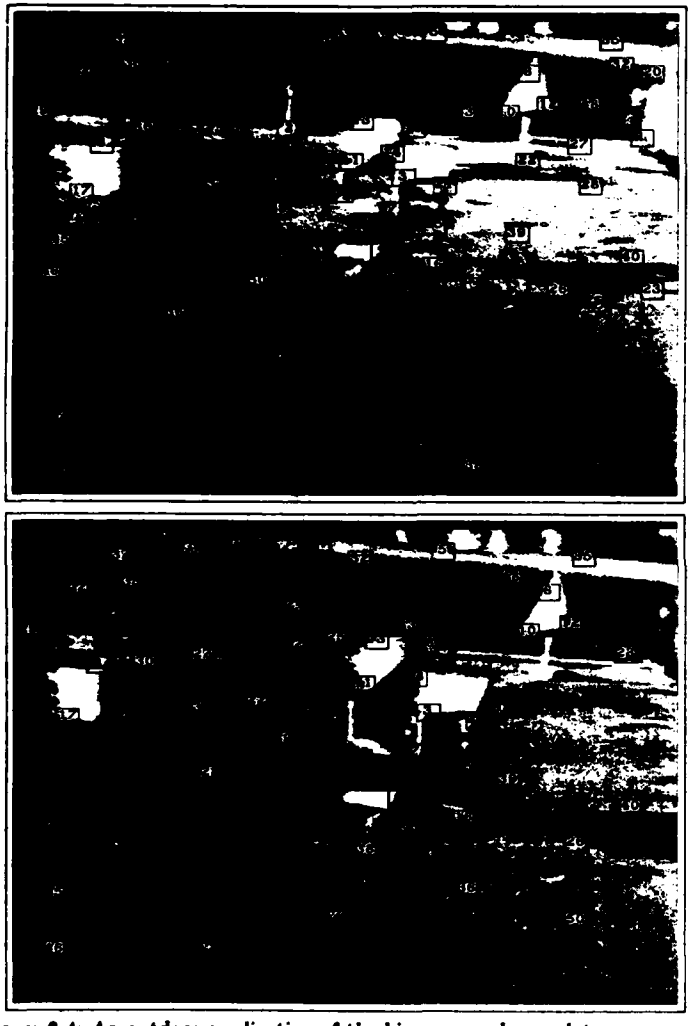


Figure 6-4: *An outdoor application of the binary search correlator.*

Each pixel comparison takes about one microsecond. The time required to locate an 8 by 8 window in a 16 by 16 search area is about 10 milliseconds. A single feature requires 5 such searches, for a total per feature time of 50 ms.

One of the three instructions could be eliminated if $\sum B$ and $\sum B^2$ were precomputed for every position in the picture. This can be done incrementally, involving examination of each pixel only twice, and would result in an overall speedup if many features are to be searched for in the same general area.

The correlator has approximately a 10% error rate on features selected by the interest operator in our sample pictures. Typical image pairs are generally taken about two feet apart with a 60° field of view camera.

Stereo

Slider Stereo

At each pause on its computer controlled itinerary the cart slides its camera from left to right on the 52 cm track, taking 9 pictures at precise 6.5 cm intervals.

Points are chosen in the fifth (middle) of these 9 images, either by the correlator to match features from previous positions, or by the interest operator.

The camera slides parallel to the horizontal axis of the (distortion corrected) camera co-ordinate system, so the parallax induced apparent displacement of features from frame to frame in the 9 pictures is purely in the X direction.

The correlator looks for the points chosen in the central image in each of the eight other pictures. The search is restricted to a narrow horizontal band. This has little effect on the computation time, but it reduces the probability of incorrect matches.

In the case of correct matches, the distance to the feature is inversely proportional to its displacement from one image to another. The uncertainty in such a measurement is the difference in distance a shift one pixel in the image

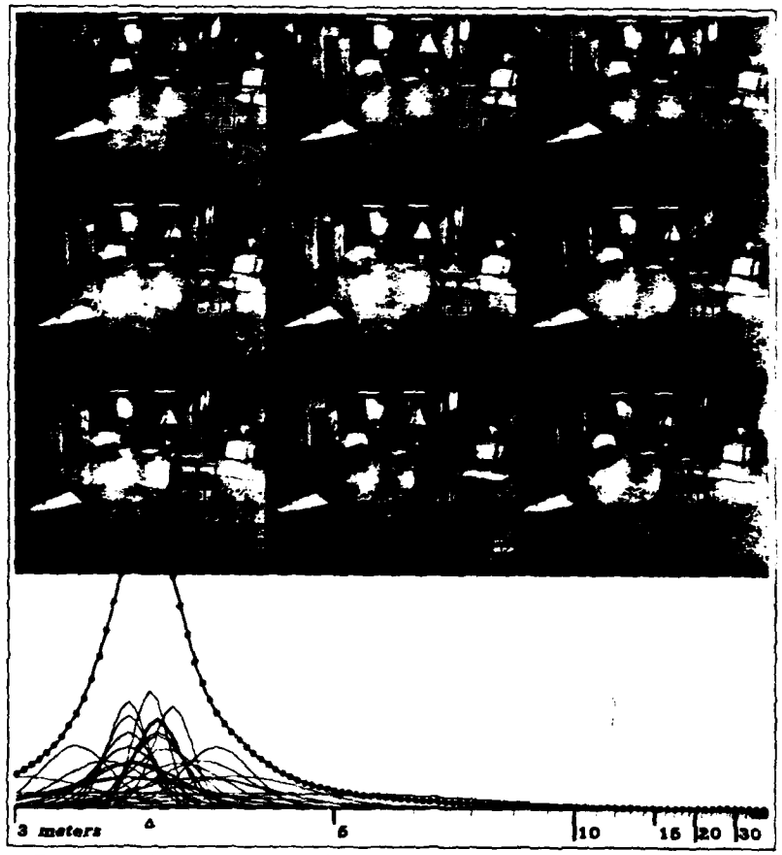


Figure 7-1: *A typical ranging. The nine pictures are from a slider scan. The interest operator chose the marked feature in the central image, and the correlator found it in the other eight. The small curves at bottom are distance measurements of the feature made from pairs of the images. The large beaded curve is the sum of the measurements over all 36 pairings. The horizontal scale is linear in inverse distance.*

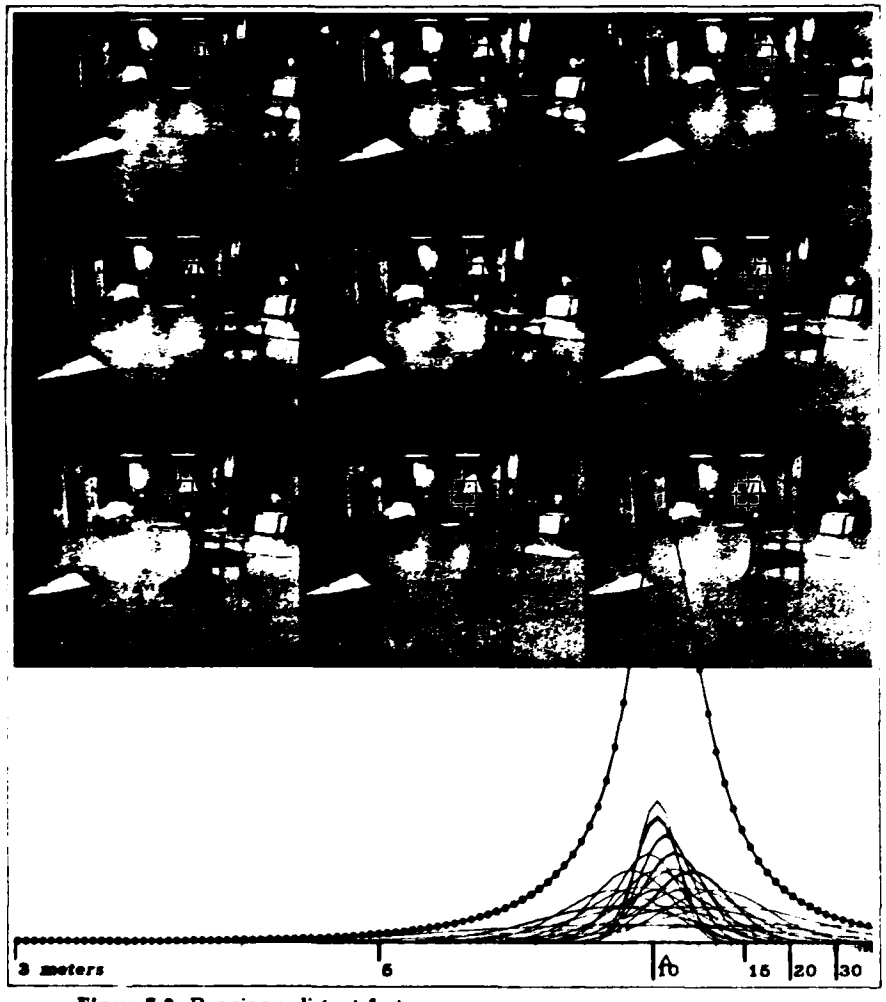


Figure 7-2: *Ranging a distant feature*

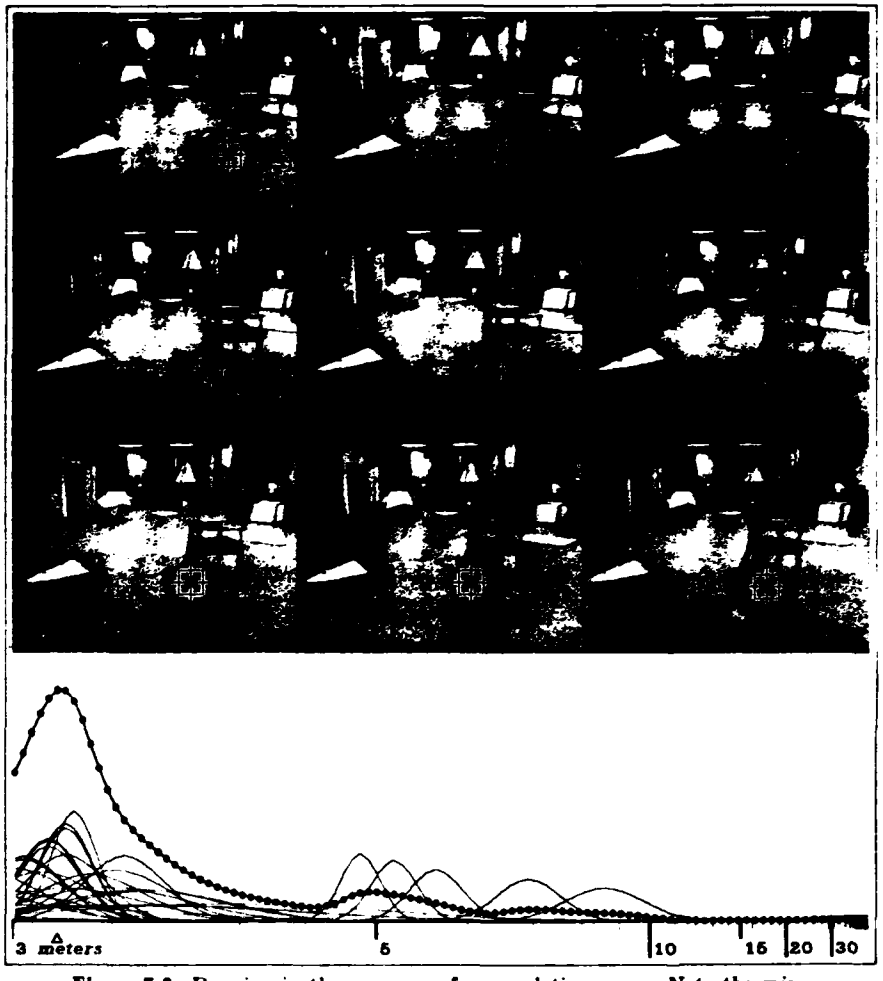


Figure 7-3: *Ranging in the presence of a correlation error. Note the mismatch in the last image.*

would make. The uncertainty varies inversely with the physical separation of the camera positions where the pictures were taken (the stereo baseline). Long baselines give more accurate distance measurements..

After the correlation step the program knows a feature's position in nine images. It considers each of the 36 ($= \binom{9}{2}$) possible image pairings as a stereo baseline, and records the estimated distance to the feature (actually inverse distance) in a histogram. Each measurement adds a little normal curve to the histogram, with mean at the estimated distance, and standard deviation inversely proportional to the baseline, reflecting the uncertainty. The area under each curve is made proportional to the product of the correlation coefficients of the matches in the two images (in central image this coefficient is taken as unity), reflecting the confidence that the correlations were correct. The area is also scaled by the normalised dot products of X axis and the shift of the features in each of the two baseline images from the central image. That is, a distance measurement is penalised if there is significant motion of the feature in the Y direction.

The distance to the feature is indicated by the largest peak in the resulting histogram, if this peak is above a certain threshold. If below, the feature is forgotten about.

The correlator frequently matches features incorrectly. The distance measurements from incorrect matches in different pictures are usually inconsistent. When the normal curves from 36 pictures pairs are added up, the correct matches agree with each other, and build up a large peak in the histogram, while incorrect matches spread themselves more thinly. Two or three correct correlations out of the eight will usually build a peak sufficient to offset a larger number of errors.

In this way eight applications of a mildly reliable operator interact to make a very reliable distance measurement. Figures 7-1 through 7-3 show typical rangings. The small curves are measurements from individual picture pairs, the beaded curve is the final histogram.

Motion Stereo

The cart navigates exclusively by vision. It deduces its own motion from the apparent 3D shift of the features around it.

After having determined the 3D location of objects at one position, the computer drives the cart about a meter forward.

At the new position it slides the camera and takes nine pictures. The correlator is applied in an attempt to find all the features successfully located at the previous position. Feature descriptions extracted from the central image at the last position are searched for in the central image at the new stopping place.

Slider stereo then determines the distance of the features so found from the cart's new position. The program now knows the 3D position of the features relative to its camera at the old and the new locations. It can figure out its own movement by finding the 3D co-ordinate transform that relates the two.

There can be mis-matches in the correlations between the central images at two positions and, in spite of the eight way redundancy, the slider distance measurements are sometimes in error. Before the cart motion is deduced, the feature positions are checked for consistency. Although it doesn't yet have the co-ordinate transform between the old and new camera systems, the program knows

the distance between pairs of positions should be the same in both. It makes a matrix in which element $[i, j]$ is the absolute value of the difference in distances between points i and j in the first and second co-ordinate systems divided by the expected error (based on the one pixel uncertainty of the ranging).

Each row of this matrix is summed, giving an indication of how much each point disagrees with the other points. The idea is that while points in error disagree with virtually all points, correct positions agree with all the other correct ones, and disagree only with the bad ones.

The worst point is deleted, and its effect is removed from the remaining points in the row sums. This pruning is repeated until the worst error is within the error expected from the ranging uncertainty.

After the pruning, the program has a number of points, typically 10 to 20, whose position error is small and pretty well known. The program trusts these, and records them in its world model, unless it had already done so at a previous position. The pruned points are forgotten forevermore.

Now comes the co-ordinate transform determining step. We need to find a three dimensional rotation and translation that, if applied to the co-ordinates of the features at the first position, minimizes the sum of the squares of the distances between the transformed first co-ordinates and the raw co-ordinates of the corresponding points at the second position. Actually the quantity that's minimized is the foregoing sum, but with each term divided by the square of the uncertainty in the 3D position of the points involved, as deduced from the one pixel shift rule. This weighting does not make the solution more difficult.

The error expression is expanded. It becomes a function of the rotation

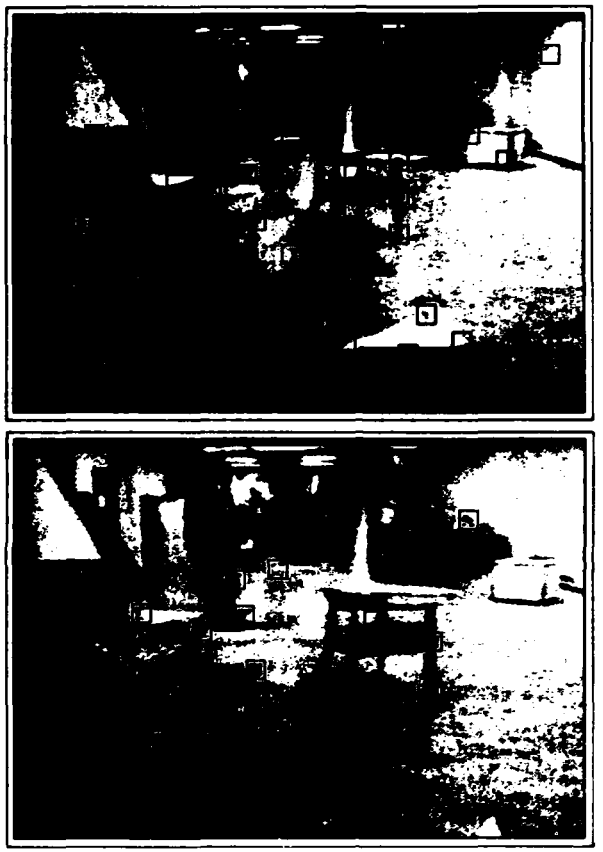


Figure 7-4: *The feature list before and after the mutual-distance pruning step. In this diagram the boxes represent features whose three dimensional position is known.*

and translation, with parameters that are the weighted averages of the x, y and z co-ordinates of the features at the two positions, and averages of their various

Figure 7-5: *Another pruning example, in more difficult circumstances. Sometimes the pruning removed too many points. The cart collided with the cardboard tree to the left later in this run.*

cross-products. These averages need to be determined only once, at the begining of the transform finding process.

To minimize the error expression, its partial derivative with respect to each variable is set to zero. It is relatively easy to simultaneously solve the three linear equations thus resulting from the vector offset, getting the optimal offset values for a general rotation. This gives symbolic expressions (linear combinations of the rotation matrix coefficients) for each of the three vector components. Substituting these values into the error expression makes it a function of the rotation alone. This new, translation determined, error expression is used in all the subsequent steps.

Minimizing the error expression under rotation is surprisingly difficult, mainly because of the non-linear constraints in the 3D rotation matrix. The next six paragraphs outline the struggle. Each step was forced by the inadequacies of the previous one.

The program begins by ignoring the non-linearities. It solves for the general 3D linear transformation, nine elements of a matrix, that minimizes the least square error. The derivatives of the error expression with respect to each of the matrix coefficients are equated to zero, and the nine resulting simultaneous linear equations are solved for the nine coefficients. If the points had undergone an error-free rigid rotation and translation between the two positions, the result would be the desired rotation matrix, and the problem would be solved.

Because there are errors in the determined position of the features, the resulting matrix is usually not simply a rotation, but involves stretching and skewing. The program ortho-normalizes the matrix. If the position errors were sufficiently small, this new matrix would be our answer.

The errors are high enough to warrant adding the rigid rotation constraints in the least squares minimization. The error expression is converted from a linear

expression in nine matrix coefficients into an unavoidably non-linear function in three parameters that uniquely characterise a rotation.

This new error expression is differentiated with respect to each of the three rotation parameters, and the resulting expressions are equated to zero, giving us three non-linear equations in three unknowns. A strenuous attempt at an analytic solution of this simultaneous non-linear system failed, so the program contains code to solve the problem iteratively, by Newton's method.

The rotation expressed by the ortho-normalised matrix from the previous step becomes the initial approximation. Newton's method for a multi-variate system involves finding the partial derivative of each expression whose root is sought with respect to each variable. In our case there are three variables and three equations, and consequently nine such derivatives. The nine derivatives, each a closed form expression of the rotation variables, are the coefficients of a 3 by 3 covariance matrix that characterises the first order changes in the expressions whose roots are sought with the parameters. The next Newton's method approximation is found by multiplying the inverse of this matrix by the value of the root expressions, and subtracting the resulting values (which will be 0 at the root) from the parameter values of the previous approximation.

Four or five iterations usually brings the parameters to within our floating point accuracy of the correct values. Occasionally, when the errors in the determined feature locations are high, the process does not converge. The program detects this by noting the change in the original error expression from iteration to iteration. In case of non-convergence, the program picks a random rotation as a new starting point, and tries again. It is willing to try up to several hundred times. The rotation with the smallest error expression ever encountered during

such a search (including the initial approximation) is returned as the answer.

Since the summations over the co-ordinate cross-products are done once and for all at the begining of the transformation determination, each iteration, involving evaluation of about a dozen moderately large expressions and a 3 by 3 matrix inversion, is relatively fast. The whole solving process, even in cases of pathological non-convergence, takes one or two seconds of computer time.

Appendix 7 presents the mathematics of the transform finder in greater detail.

Path Planning

The cart vision system has an extremely simple minded approach to the world. It models everything it sees as clusters of points. If enough such points are found on each nearby object, this model is adequate for planning a non-colliding path to a destination.

The features in the cart's 3D world model can be thought of as fuzzy ellipsoids, whose dimensions reflect the program's uncertainty of their position. Repeated applications of the interest operator as the cart moves cause virtually all visible objects to be become modelled as clusters of overlapping ellipsoids.

To simplify the problem, the ellipsoids are approximated by spheres. Those spheres sufficiently above the floor and below the cart's maximum height are projected on the floor as circles. The cart itself is modelled as a 3 meter circle. The path finding problem then becomes one of maneuvering the cart's 3 meter circle between the (usually smaller) circles of the potential obstacles to a desired location.

It is convenient (and equivalent) to conceptually shrink the cart to a point, and add its radius to each and every obstacle. An optimum path in this environment will consist of either a straight run between start and finish, or a series

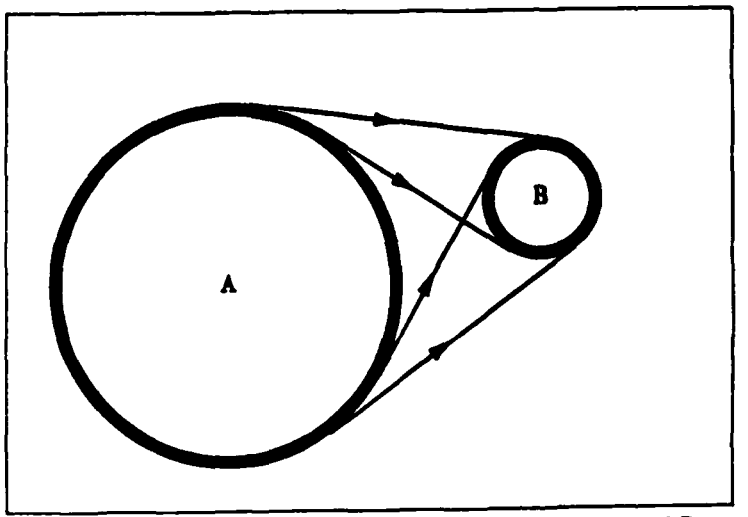


Figure 8-1: *The four tangential paths between circular obstacles A and B*

of tangential segments between the circles and contacting arcs (imagine loosely laying a string from start to finish between the circles, then pulling it tight).

Superficially, the problem seems to be one of finding the shortest path in a graph of connected vertices. The tangential segments are the edges of the graph, the obstacles, along with the destination and source, are the vertices. There are algorithms (essentially breadth first searches, that repeatedly extend the shortest path to any destination encountered) which, given the graph, can find the desired path in $O(n^2)$ time, where n is the number of vertices. On closer inspection, a few complications arise when we try to apply such an algorithm.

There are four possible paths between each pair of obstacles (Figure 8-1). because each tangent can approach clockwise or counterclockwise. Expanding

each obstacle into two distinct vertices, one for clockwise circumnavigations, the other for counterclockwise paths, handles this.

Setting up the distance matrix of the graph involves detecting which of the tangential paths are not allowed, because they blocked by other obstacles (such blocked paths are represented by infinite distances). There are $O(n^2)$ tangent paths between obstacle pairs. Determining whether each particular path is blocked involves examining at least a *fraction* of the other obstacles, a process that takes $O(n)$ time. Thus generating the distance graph, whether explicitly before running the shortest path algorithm, or implicitly within the algorithm itself, takes $O(n^3)$ time. With this consideration, the algorithm is $O(n^3)$.

The obstacles are not dimensionless points. Arriving on one tangent and leaving on another also involves travel on the circular arc between the tangents. Furthermore, paths arriving at an obstacle tangentially from different places do not end up at the same place. Our circular obstacles occupy a finite amount of space. Both these considerations can be handled by noting that there are only a finite number of tangent points around each obstacle we need consider, and these tangent points are dimensionless.

Each obstacle develops four tangent points because of the existence of every other obstacle. A path problem with n circular obstacles can thus be translated exactly into a shortest path in graph problem with $4n(n-1)$ vertices, each edge in the graph corresponding to a tangent between two obstacles plus the arc leading from one end of the tangent path to the beginning of another one. The solution time thus appears to grow to $O(n^4)$. Fundamentally, this is correct, but significant shortcuts are possible.

The distance matrix for the tangent points is extremely sparse. In our possible solution space, each tangent point leading from an obstacle connects to only about $2n$ others, out of the $4n(n-1)$ possible. This fact can be used to reduce the amount of work from $O(n^4)$ to about $O(n^3)$. Appendix 8 gives the details.

The algorithm just outlined finds the guaranteed shortest obstacle avoiding path from start to finish. It is rather expensive in time, and especially in space. It requires several two dimensional arrays of size n by n. The number of obstacles sometimes grows to be about 100. Because both storage and running time needed conservation, the final version of the cart program used a simplified,

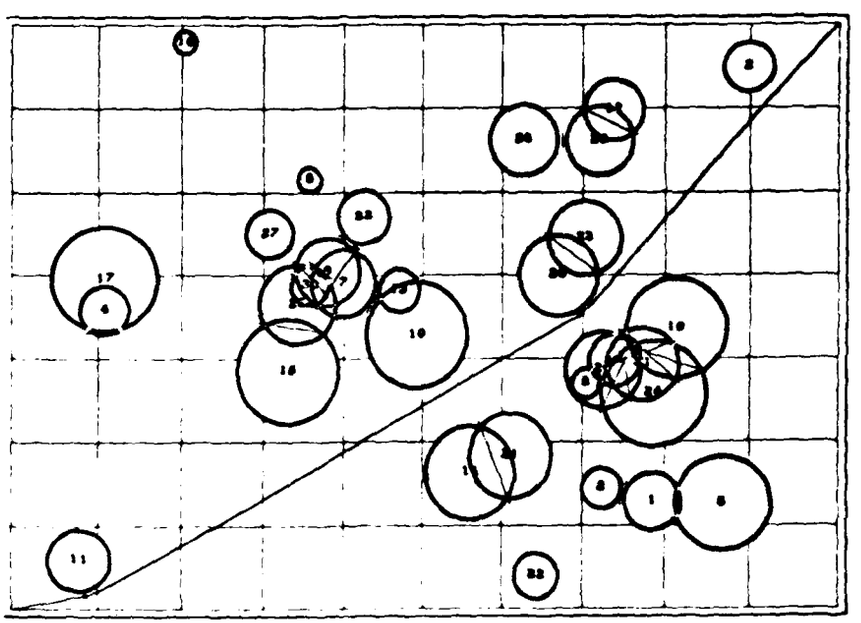


Figure 8-2: The shortest path finder's solution to a randomly constructed problem. The route is from the lower left corner to the uper right. The numbered circles are the obstacles, the wiggly line is the solution.

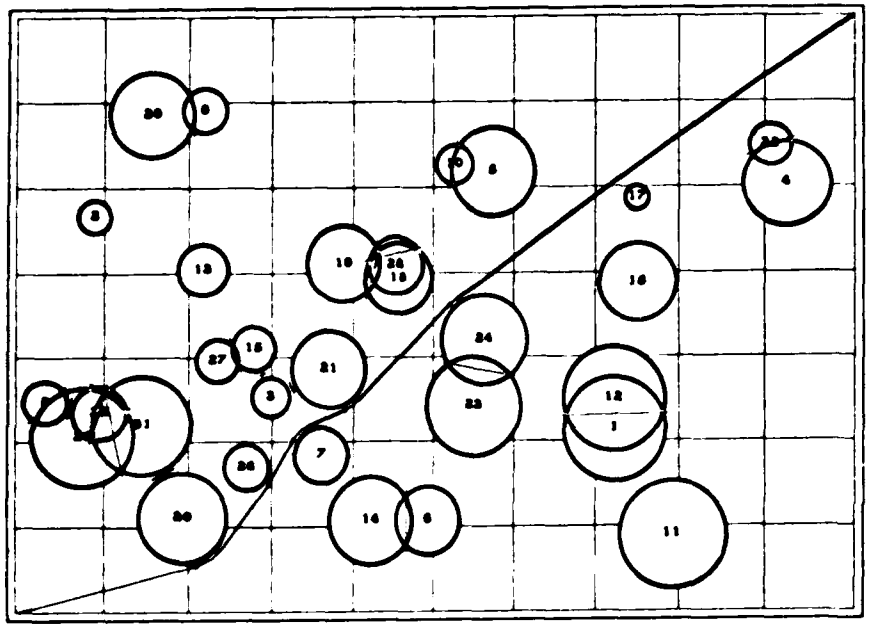


Figure 8-3: *Another path finder solution.*

and considerably cheaper, approximation to this approach.

The simplified program, also described in greater detail in Appendix 8, does not distinguish between different tangent points arriving at a single obstacle. Instead of a very sparse distance matrix of size $4n(n-1)$ squared, it deals with a dense matrix of dimension $2n$ by $2n$. Many of the arrays that were of size n^2 in the full algorithm are only of dimension n in the cheap version. The arc lengths for travel between tangents are added into the computed distances, but sometimes too late to affect the search. If the obstacles were all of zero radius, this simple algorithm would still give an exact solution. As obstacle size grows, so does the

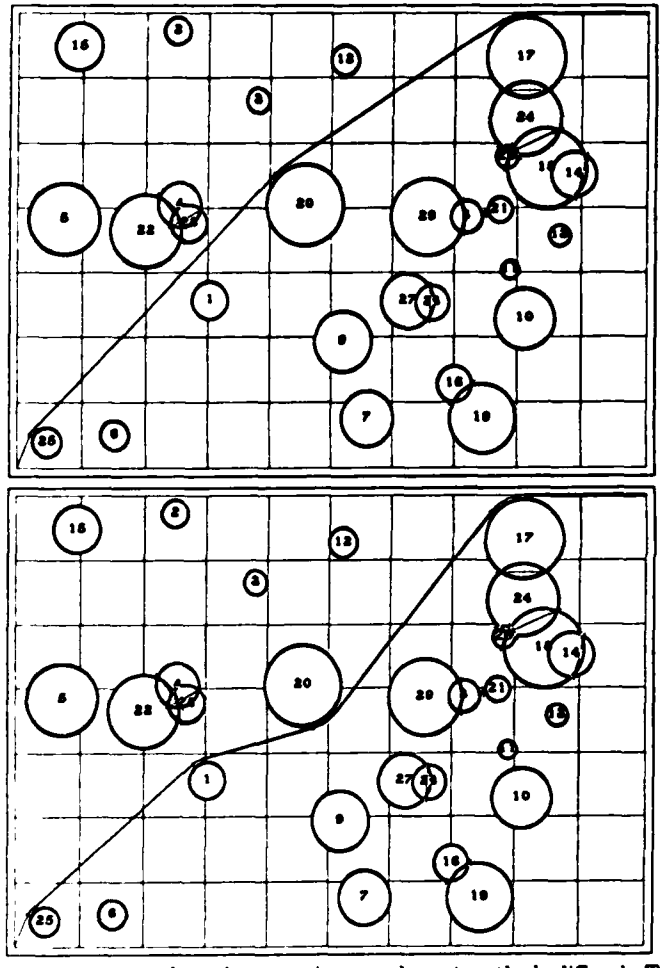


Figure 8-4: *A case where the approximate and exact methods differed. Top diagram is the exact solution, bottom one is the approximate algorithm's guess.*

probability of non-optimal solutions.

In randomly generated test cases containing about fifty typical obstacles, the approximation finds the best solution about 90% of the time. In the other cases it produces solutions only slightly longer.

A few other considerations are essential in the path planning. The charted routes consist of straight lines connected by tangent arcs, and are thus plausible paths for the cart, which steers like an automobile. This plausibility is not necessarily true of the start of the planned route, which, as presented thus far, does not take the initial heading of the cart into account. The plan could, for instance, include an initial segment going off 90° from the direction in which the cart points, and thus be impossible to execute.

The current code handles this problem by including a pair of "phantom" obstacles along with the real perceived ones. The phantom obstacles have a radius equal to the cart's minimum steering radius, and are placed, in the planning process, on either side of the cart at such a distance that after their radius is augmented by the cart's radius (as happens for all the obstacles), they just touch the cart's centroid, and each other, with their common tangents being parallel to the direction of the cart's heading. They effectively block the area made inaccessible to the cart by its maneuverability limitations.

In the current program the ground plane, necessary to decide which features are obstacles, and which are not, is defined a priori, from the known height of the cart camera above the floor, and the angle of the camera with respect to the horizontal (measured before a run by a protractor/level). Because the program runs so slowly that the longest feasible travel distance is about 20 meters, this is adequate for now. In later, future, versions the cart should dynamically update its

ground plane orientation model by observing its own motion as it drives forward. The endpoints of each meter-long lurch define a straight line that is parallel to the local ground. The vector component of the ground plane model in the direction of the lurch can be tilted to match the observed cart motion, while the component perpendicular to that is left unchanged. After moving in two non-colinear lurches, all ground-plane orientation parameters would be updated. This process would allow the cart to keep its sanity while traversing hilly terrain. Because the motion determination has short term inaccuracies, the tilt model should be updated only fractionally at each move, in the manner of exponential smoothing.

Path Execution

After the path to the destination has been chosen, a portion of it must be implemented as steering and motor commands and transmitted to the cart. The control system is primitive. The drive motor and steering motors may be turned on and off at any time, but there exists no means to accurately determine just how fast or how far they have gone. The current program makes the best of this bad situation by incorporating a model of the cart that mimics, as accurately as possible, the cart's actual behavior. Under good conditions, as accurately as possible means about 20%; the cart is not very repeatable, and is affected by ground slope and texture, battery voltage, and other less obvious externals.

The path executing routine begins by excising the first .75 meters of the planned path. This distance was chosen as a compromise between average cart velocity, and continuity between picture sets. If the cart moves too far between picture digitizing sessions, the picture will change too much for reliable correlations. This is especially true if the cart turns (steers) as it moves. The image seen

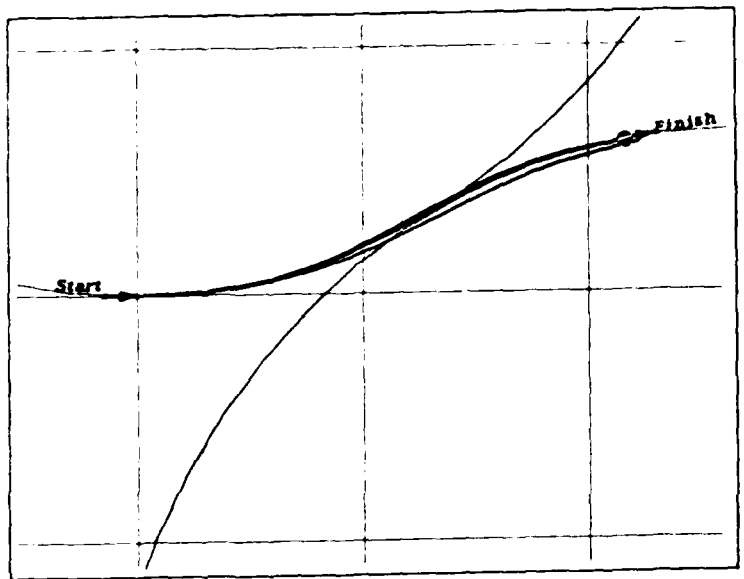


Figure 8-5: *An example of the simulator's behavior. The diagram is a plan view of the path executer's world model; the grid cells are one meter on a side. The cart's starting position and final destination and orientation are indicated by arrows. The two large circles, only portions of which are visible, represent the analytic two-arc path. It goes from Start through the tangent of the two circles to Finish. The heavier paths between the two points represent the iterations of the simulator as its parameters were adjusted to compensate for the cart's dynamic response.*

by the camera then pans across the field of view. The cart has a wide angle lens that covers 60° horizontally. The .75 meters, combined with the turning radius limit (5 meters) of the cart results in a maximum shift in the field of view of 15°, one quarter of the entire image.

This .75 meter segment can't be followed precisely, in general, because of

dynamic limits in the cart motion. The cart can steer reliably only when it is driving. It takes a finite time for the steering motor to operate. When the drive motors are energized the robot takes a while to accelerate to its terminal velocity, and it coasts for a half meter when the motors are turned off. These complications were too difficult to model in the obstacle path planning.

Instead the program examines the cart's position and orientation at the end of the desired .75 meter lurch, relative to the starting position and orientation. The displacement is characterized by three parameters; displacement forward, displacement to the right and change in heading. In closed form the program computes a path that will accomplish this movement in two arcs of equal radius, but different lengths. The resulting trajectory has a general "S" shape. This closed form has three parameters; the radius of the two arcs, the distance along the first arc and the distance along the second, just the right number for a constrained solution of the desired displacement.

Making the arcs of equal radius minimises the curvature of the planned path, a desirable goal for a vehicle that steers slowly (as well as unreliably). Even with minimized curvature, the two-arc path can only be approximated, since the steering takes a finite amount of time, during which the robot must be rolling.

I was unable to find a closed form expressing the result of simultaneous steering and driving, so the program relies on a simulation. The on and off times for the drive motor necessary to cause the cart to cover the required distance are computed analytically, as are the steering motor on times necessary to set the cart turning with the correct radii. These timings are then fed to the simulator and the final position of the cart is examined. Because the steering was not instantaneous, the simulated path usually turns out to be less curvy than the requested

one. The difference between the simulated final position and orientation and the desired one is used to generate a new input for the analytic solver (To clarify; if the simulation says the cart ends up one meter too far to the right, the next iteration will request a position one meter leftward. This process works well when the results of the simulation react nearly linearly to the initial requests). About five iterations of this step are usually sufficient to find an adequate command sequence. This sequence is then transmitted, and the cart moves, more or less as simulated.

Except for the endpoints, the path generated in this way differs, in general, from the one produced by the obstacle avoider algorithm. For .75 meter lurches, however, it stays within a few centimeters of it. The cart avoids each obstacle by a safety factor of about a half meter, so such inaccuracies can be tolerated. In any case, the mechanical precision of the cart's response is poor enough, and its seeing sparse enough, to require such a safety margin.

Evaluation

Many years ago I chose the line of research described herein intending to produce a combination of hardware and software by which the cart could visually navigate reliably in most environments. For a number of reasons, the existing system is only a first approximation to that youthful ideal.

One of the most serious limitations is the excruciating slowness of the program. In spite of my best efforts, and many compromises, in the interest of speed, it takes 10 to 15 minutes of real time to acquire and consider the images at each lurch, on a lightly loaded KL-10. This translates to an effective cart velocity of 3 to 5 meters an hour. Interesting obstacle courses (2 or three major obstacles, spaced far enough apart to permit passage within the limits of the cart's size and maneuverability) are at least 15 meters long, so interesting cart runs take from 3 to 5 hours, with little competition from other users, impossibly long under other conditions.

During the last few weeks of the AI lab's residence in the D.C. Power building, when the full fledged obstacle runs described here were executed, such conditions of light load were available on only some nights, between 2 and 6 AM and on some weekend mornings. The cart's video system battery lifetime on a full charge is at most 5 hours, so the limits on field tests, and consequently on

the debug/improve loop, were strictly circumscribed.

Although major portions of the program had existed and been debugged for several years, the complete obstacle avoiding system (including fully working hardware, as well as programs) was not ready until two weeks before the lab's scheduled move. The first week was spent quashing unexpected trivial bugs, causing very silly cart behavior under various conditions, in the newest parts of the code, and recalibrating camera and motor response models.

The final week was devoted to serious observation (and filming) of obstacle runs. Three full (about 20 meter) runs were completed, two indoors and one outdoors. Two indoor false starts, aborted by failure of the program to perceive an obstacle, were also recorded. The two long indoor runs were nearly perfect.

In the first, the cart successfully slalomed its way around a chair, a large cardboard icosahedron, and a cardboard tree then, at a distance of about 16 meters, encountered a cluttered wall and backed up several times trying to find a way around it.

The second indoor run involved a more complicated set of obstacles, arranged primarily into two overlapping rows blocking the goal. The cart backed up twice to negotiate the tight turn required to go around the first row, then executed several steer forward / back up moves, lining itself up to go through a gap barely wide enough in the second row. This run had to be terminated, sadly, before the cart had gone through the gap because of declining battery charge and increasing system load.

The outdoor run was less successful. It began well; in the first few moves the program correctly perceived a chair directly in front of the camera, and a

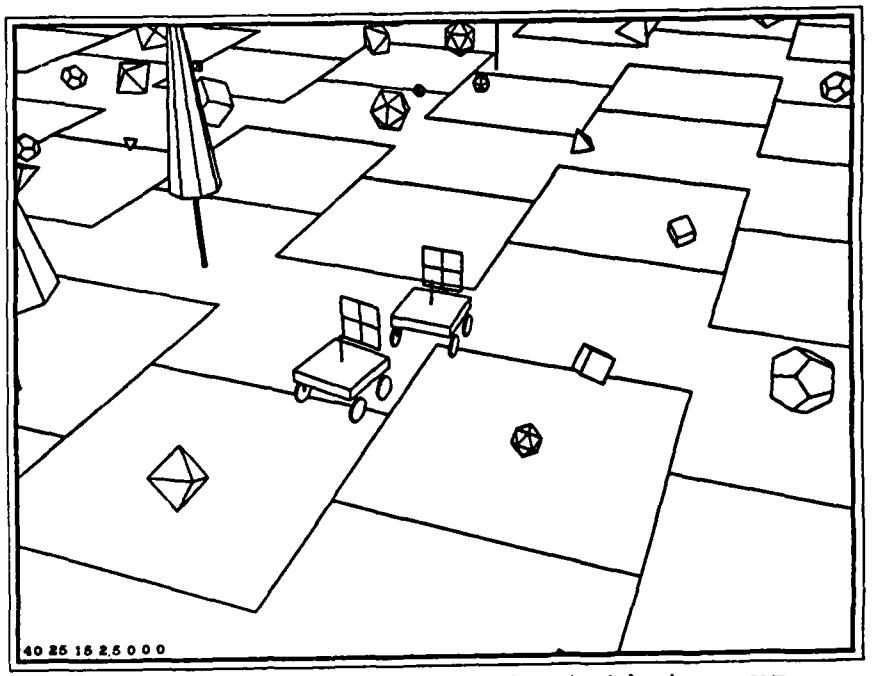


Figure 9-1: *A sample output from the three dimensional drawing program that inspired the construction of the ill fated cardboard trees and rocks.*

number of more distant cardboard obstacles and sundry debris. Unfortunately, the program's idea of the cart's own position became increasingly wrong. At almost every lurch, the position solver deduced a cart motion considerably smaller than the actual move. By the time the cart had rounded the foreground chair, its position model was so far off that the distant obstacles were replicated in different positions in the cart's confused world model, because they had been seen early in the run and again later, to the point where the program thought an actually

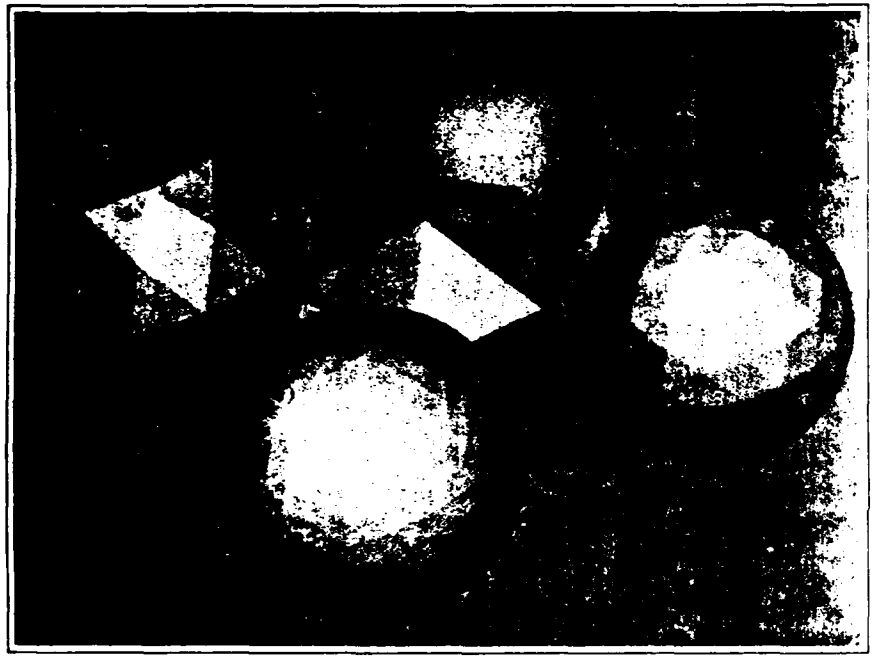


Figure 9-2: *Gray scale output from the 3D program. See how seductive the pictures are?*

existing distant clear path was blocked. I restarted the program to clear out the world model when the planned path became too silly. At that time the cart was four meters in front of a cardboard icosahedron, and its planned path lead straight through it. The newly re-incarnated program failed to notice the obstacle, and the cart collided with it. I manually moved the icosahedron out of the way, and allowed the run to continue. It did so uneventfully, though there were continued occasional slight errors in the self position deductions. The cart encountered a large cardboard tree towards the end of this journey and detected a portion of it

only just in time to squeak by without colliding.

The two short abortive indoor runs involved setups nearly identical to the two-row successful long run described one paragraph ago. The first row, about three meters in front of the cart's starting position contained a chair, a real tree (a small cypress in a planting pot), and a polygonal cardboard tree. The cart saw the chair instantly and the real tree after the second move, but failed to see the cardboard tree ever. Its planned path around the two obstacles it did see put it on a collision course with the unseen one. Placing a chair just ahead of the cardboard tree fixed the problem, and resulted in a successful run. Never, in all my experience, has the code described in this thesis failed to notice a chair in front of the cart.

Flaws Found

These runs suggest that the system suffers from two serious weaknesses. It does not see simple polygonal (bland and featureless) objects reliably, and its visual navigation is fragile under certain conditions. Examination of the program's internal workings suggests some causes and possible solutions.

Bland Interiors

The program sometimes fails to see obstacles lacking sufficient high contrast detail within their outlines. In this regard, the polygonal tree and rock obstacles I whimsically constructed to match diagrams from a 3D drawing program, were a terrible mistake. In none of the test runs did the programs ever fail to see a chair placed in front of the cart, but half the time they did fail to see a pyramidal tree

or an icosahedral rock made of clean white cardboard. These contrived obstacles were picked up reliably at a distance of 10 to 15 meters, silhouetted against a relatively unmoving (over slider travel and cart lurches) background, but were only rarely and sparsely seen at closer range, when their outlines were confused by a rapidly shifting background, and their bland interiors provided no purchase for the interest operator or correlator. Even when the artificial obstacles were correctly perceived, it wa. by virtue of only two to four features. In contrast, the program usually tracked five to ten features on nearby chairs.

It may seem ironic that my program does poorly in the very situations that were the only possible environment for one of its predecessors, SRI's Shakey. Shakey's environment was a large scale "blocks world", consisting entirely of simple, uniformly colored prismatic solids. Its vision was edge based and monocular, except that it occasionally used a laser range finder to augment its model based 3D reasoning. My area correlation techniques were chosen to work in highly complex and textured "real world" surroundings. That they do poorly in blocks world contexts suggests complementarity. A combination of the two might do better than either alone.

A linking edge follower could probably find the boundary of, say, a pyramidal tree in each of two disparate pictures, even if the background had shifted severely. It could do a stereo matching by noting the topological and geometric similarities between subsets of the edge lists in the two pictures. Note that this process would not be a substitute for the area correlation used in the current program, but an augmentation of it. Edge finding is expensive and not very effective in the highly textured and detailed areas that abound in the real world, and which are area correlation's forte.

Another matching method likely to be useful in some scene areas is region growing, guided by very small scale area correlation.

In the brightly sunlit outdoor run the artificial obstacles had another problem. Their white coloration turned out to be much brighter than any "naturally" occurring extended object. These super bright, glaring, surfaces severely taxed the very limited dynamic range of the cart's vidicon/digitizer combination. When the icosahedron occupied 10% of the camera's field of view, the automatic target voltage circuit in the electronics turned down the gain to a point where the background behind the icosahedron appeared nearly solid black.

Confused Maps

The second major problem exposed by the runs is glitches in the cart's self-position model. This model is updated after a lurch by finding the 3D translation and rotation that best relates the 3d position of the set of tracked features before and after the lurch. In spite of the extensive pruning that precedes this step, (and partly because of it, as is discussed later) small errors in the measured feature positions sometimes cause the solver to converge to the wrong transform, giving a position error well beyond the expected uncertainty. Features placed into the world model before and after such a glitch will not be in the correct relative positions. Often an object seen before is seen again after, now displaced, with the combination of old and new positions combining to block a path that is in actuality open.

This problem showed up mainly in the outdoor run. I've also observed it indoors in past, in simple mapping runs, before the entire obstacle avoider was

assembled. There appear to be two major causes for it, and a wide range of supporting factors.

Poor seeing, resulting in too few correct correlations between the pictures before and after a lurch, is one culprit. The highly redundant nine eyed stereo ranging is very reliable, and causes few problems, but the non-redundant correlation necessary to relate the position of features *before* and *after* a lurch, is error prone. Features which have been located in 3D *from* one picture ninetuplet are sought in the next set by applying the correlator between the central images of the two sets. The points so found are then ranged using nine eyed stereo in the new picture set. The cart's motion is deduced by finding the apparent 3D movement of the features from one picture set to the next.

Before this 3D co-ordinate transformation is computed, the matched points are pruned by considering their mutual three dimensional distances in the two co-ordinate systems. Accurate to the known position uncertainty of each feature, these distances should be the same in the two systems. Points that disagree in this measure with the majority of other points are rejected.

If too few points are *correctly* matched, because the seeing was poor, or the scene was *intrinsically* too bland, the pruning process can go awry. This happened several times in the outdoor run.

The outdoor scene was very taxing for the cart's vidicon. It consisted of large regions (mainly my cardboard constructions) glaring in direct sunlight, and other important regions in deep shadow. The color of the rest of the scene was in a relatively narrow central gray range. It proved impossible to simultaneously not saturate the glaring or the shadowed areas, and to get good contrast in the middle gray band, within the six bit (64 gray level) resolution of my digitized

pictures. To make matters even more interesting, the program ran so slowly that the shadows moved significantly (up to a half meter) between lurches. Their high contrast boundaries were favorite points for tracking, enhancing the program's confusion.

Simple Fixes

Though elaborate (and thus far untried in our context) methods such as edge matching may greatly improve the quality of automatic vision in future, subsequent experiments with the program revealed some modest incremental improvements that would have solved most of the problems in the test runs.

The issue of *unseen* cardboard obstacles turns out to be partly one of *over*-conservatism on the program's part. In all cases where the cart collided with an obstacle it had correctly ranged a few features on the obstacle in the prior nine-eyed scan. The problem was that the much more fragile correlation between vehicle forward moves failed, and the points were rejected in the mutual distance test. Overall the nine-eyed stereo produced very few errors. If the path planning stage had used the pre-pruning features (still without incorporating them permanently into the world model) the runs would have proceeded much more smoothly. All of the most vexing false negatives, in which the program failed to spot a real obstacle, would have been eliminated. There would have been a very few false positives, in which non-existent ghost obstacles would have been perceived. One or two of these might have caused an unnecessary swerve or backup. But such ghosts would not pass the pruning stage, and the run would have proceeded normally after the initial, non-catastrophic, glitch.

The self-position confusion problem is related, and in retrospect may be

considered a trivial bug. When the path planner computes a route for the cart, another subroutine takes a portion of this plan and implements it as a sequence of commands to be transmitted to the cart's steering and drive motors. During this process it runs a simulation that models the cart acceleration, rate of turning and so on, and which provides a prediction of the cart's position after the move. With the current hardware the accuracy of this prediction is not great, but it nevertheless provides much a priori information about the cart's new position. This information is used, appropriately weighted, in the least-squares co-ordinate system solver that deduces the cart's movement from the apparent motion in 3D of tracked features. It is not used, however, in the mutual distance pruning step that preceeds this solving. When the majority of features have been correctly tracked, failure to use this information does not hurt the pruning. But when the seeing is poor, it can make the difference between choosing a spuriously agreeing set of mis-tracked features and the small correctly matched set.

Incorporating the prediction into the pruning, by means of a heavily weighted point that the program treats like another tracked feature, removes almost all the positioning glitches when the program is fed the pictures from the outdoor run.

I have not attempted any live cart runs with these program changes because the cramped conditions in our new on-campus quarters make cart operations nearly impossible.

CHAPTER 10

Spinoffs

Graphics

The display hardware in the early days of the AI lab was strictly vector oriented; six vector terminals from Information International Inc., and a Calcomp plotter. When I arrived at Stanford the lab had just acquired a new raster based display system from Data Disc Corp. The display packages in existence at the time, of which there were several, had all started life in the vector environment, and were all oriented around vector list display descriptions. Some of the packages had been extended by addition of routines that scanned such a vector list and drew the appropriate lines in a Data Disc raster.

In my opinion, this approach had two drawbacks if raster displaying was to supplant vector drawing. The vector list is compact for simple pictures, but can grow arbitrarily large for complex pictures with many lines. A pure raster representation, on the other hand, needs a fixed amount of storage for a given raster size, independent of the complexity of the image in the array. I often saw programs using the old display packages bomb when the storage allocated for their display lists was exceeded. A second objection to vector list representations is that they had no elegant representation for some of the capabilities of raster

devices not shared by vector displays, notably large filled in areas.

These thoughts prompted me to write a raster oriented display package for the Data Disc monitors that included such primitives as ellipse and polygon filling (including convex and star polygons), darkening and inversion as well as lighting of filled areas, in addition to the traditional linear commands. This package has developed a large following, and has been translated into several languages (the original package was written to run in a SAIL environment. Portions of it have been modified to run in raw assembly code, and under Lisp 1.6 [W2] and Maclisp [B1] [L1]). It is outlined in Appendix 10.

The Data Disc package was built around a rather peculiar and inflexible raster format (for instance, the raster lines are four-way interleaved) made necessary by the nature of the Data Disc hardware. When our XGP arrived there was no easy way to extend it to handle buffers for both the Data Disc and the much higher resolution XGP, though I did add a small routine which produced a coarse XGP page by expanding each Data Disc raster pixel.

Thus the arrival of the XGP created the need for a new package which could generate high resolution XGP images from calls similar to the ones used with Data Disc. I fashioned one by modifying the innermost parts of a copy of the older routines.

New raster devices were appearing,and a single system able to handle all of them became clearly desirable. The general byte raster format used in the vision package described later in this chapter was a good medium for implementing such generality, and I once again made a modified version of the Data Disc package which this time drew into arbitrary rectangular subwindows of arbitrarily sized bit rasters. I added a few new features, such as the ability to deposit characters

from XGP font files and halftone pictures into the drawing. Only partially implemented at this writing is a package which draws into byte rasters, using gray scale to produce images with reduced edge jagginess.

With both hard and soft copy graphic output devices available it became

Figure 10-1: *A silly picture produced from a GOD file. The little program which wrote the file may be found in Appendix 10.*

desirable to write programs which drew the same picture into buffers of different resolutions destined for different devices. Since the buffers were sometimes very large, it was reasonable to put them in separate core images. Thus evolved a message based version of the graphics routines in which the graphics main program does none of the actual drawing itself, but creates graphic "slaves" which run as separate jobs, and to which it sends messages such as "draw a line from [1.5, 2.3] to [5.2, 0.1]." A large number of such graphic servers can exist at any one time, and they can be individually activated and deactivated at any time. Whenever the controlling program executes a graphics primitive, all the servers currently active do the appropriate drawing. The messages sent to graphics servers can also be written into files (which can be created, destroyed, activated and deactivated just as if they were servers). Such files, which I call GOD files (for Graphics On Displays, Documents and other Devices), can be saved and used to generate drawings off-line, by manually feeding them to any of the available graphics servers. GOD files can also be used as subroutines in later drawing programs, and in more powerful ways, as suggested in the next paragraph.

A version of these routines is at the core of XGPSYN and XGPSYG, programs which can display pages from XGP multifonted documentation files readably as gray scale images on standard TV monitors, as well as being able to list them on the XGP, after composing them as full bit rasters. XGPSYG, additionally, can insert diagrams and halftone representations of gray scale pictures into the documents being printed, in response to escape sequences occurring in the documents, pointing to GOD graphic files, hand eye format picture files or Leland Smith's music manuscript plot files (!).

The obstacle avoider program used the message graphics system to document its internal state, generating in the process some of the diagrams seen in this

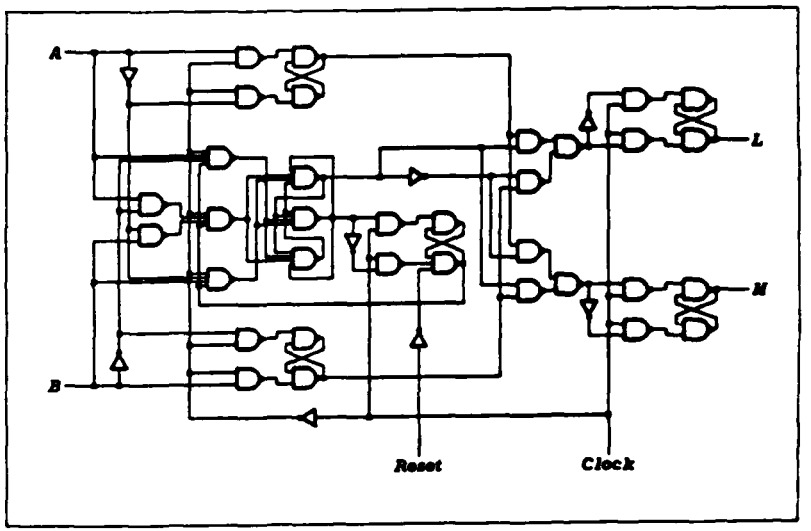


Figure 10-2: *Another GOD file example. This diagram was produced with the help of a circuit drawing extension to the main package.*

thesis. The text in the thesis was formatted with Don Knuth's TEX typesetting system [K1], and printed with diagrams using XGPSYG.

Vision Software

The Hand-Eye project had collected a moderately large package of utility subroutines for acquiring and analysing pictures when I began my vision work. Unfortunately it was far from complete, even in its basics, and was built around a "global" picture representation (global variables held picture size and location parameters) which made dealing with several pictures at the same time nearly impossible, especially if they were of different sizes. This was a great handicap to

me since I worked nearly from the start with hierarchies of reduced pictures, and with excerpted windows. The format also made certain primitive operations, such as individual byte accessing, unnecessarily difficult because it carried insufficient precomputed data.

In my opinion there was little worth salvaging, so I began writing my own vision primitives from scratch, starting with a data representation that included with each picture constants such as the number of words in a scanline, as well as a table of byte pointer skeletons for accessing each column. It features high speed in the utility operations, and in such fancier things as correlation and filtering. It has a clever digitising subroutine that compensates for some of the limitations of our hardware. This package has grown over the years and is now considerably more extensive than the Hand Eye library, and has largely supplanted it in other Stanford vision projects [B2].

The obstacle avoider uses an extension of the basic package which permits convenient handling of sequences of successively reduced pictures, and of chains of windows excerpted from such sequences.

PIX is a program that uses the vision subroutines to provide desk calculator type services for pictures to a user. It can digitize pictures from various sources, transform and combine them in many ways, transfer them to and from disk, display or print them on various output devices. Among its more exotic applications has been the generation of font definitions for our printer from camera input. XGPSYN and XGPSYG also make use of the vision package. The 3D shaded graphics in this thesis were produced with a program that uses the partially implemented gray scale GOD file server, which also calls on the vision package.

Further details may be found in Appendix 10.

Future Carts

It will be obvious that the current system is very minimal, both in hardware and software. The hardware limitations not only necessitated considerable repair time, which reduced time available for more advanced work, but they also had a direct effect on the performance of the final product.

Strength of Body

Feedback to the program about steering angle and about the number of rotations of the wheels in a forward move would have permitted a much more reliable estimate of the new cart position after a move, and before the vision step. Such a *priori* estimates would have eliminated the main source of trouble in the flawed outdoor cart run, almost certainly turning it into a success in spite of the poor seeing (in the early steps of this run, the program had correctly located most of the relevant obstacles. As it proceeded forward, moving shadows and the general lack of solid features to track caused increasingly serious misestimations of its own position, and later sightings were consequently assigned the wrong positions in the world model, eventually turning it to trash).

The failure of the program to notice the fake tree in the two short abortive

indoor runs would not have been fixed by steering and travel distance feedback alone, but could have been saved if the cart also had reporting touch sensors. On running into the tree, the cart would have stopped, and its position would have been noted, along with the position of the touch relative to the body. The position of the touch would have been added to the world model, just like any visually perceived feature. The next path planning step would take it into account, causing suitable backing up of the cart when it moved, probably turning a failed run into a moderate success. Such last ditch physical obstacle detection is no substitute for improving the vision, but it would take the *system* a step closer to practicality. A proximity sensor could do even better, by detecting an imminent collision, or a near miss, without physical contact.

Sensors of this kind would require two-way data communication between the cart and its controlling computer. Currently the computer commands motions, and the cart transmits pictures only. A small computer onboard the cart would permit a bidirectional, error checked, link. It would also permit down-loading of small control programs which servoed position and speed, and had provision for dealing with unusual circumstances, such as encounters of the touch sensors with obstacles. Doing this remotely through the radio link is risky, because the data rate is not very high, and because the link is subject to noise, which could come at a critical time.

The cart's physical size, about a meter cubed, and weight, about 100 kilograms, created many experimental difficulties. It requires a lot of room to move. The outdoors is too variable an environment for serious work at the present tender stage of the software; besides, running during daylight hours is incompatible with getting many cycles out of a time-shared computer. Thus the cart needs a large indoor arena. We were lucky to have had a suitable room in the

D.C. Power building. The new computer science building is not so generously endowed. Other problems with the cart's size involve its repairability; it could not be brought into convenient shop areas since it was too wide to pass through standard size doors, and often had to be worked on in peculiar postures, since it is too heavy to turn on its side (deliberately anyway). Its weight demands large motors and batteries, and associated driver electronics; these are proportionately more expensive than smaller ones, and also harder to obtain. The large components translate to a certain coarseness in mobility and maneuverability; a reasonable itinerary must cover tens of meters. This implies that the radio links must have a useful range of hundreds of meters, to keep signal strengths roughly uniform (and to avoid having to relocate antennas when moving the cart to different nearby environments).

Progress in reducing the size and power nsumption of electronic components (particularly the existence of small solid state TV cameras), makes the concept of a much smaller robot rover, perhaps a third to one half a meter in diameter, reasonable. Instead of carrying batteries with a capacity of a thousand watt hours, it could make do with one or two hundred. Its transmitters would broadcast with fractional watts, instead of several watts, and the distances would be a few meters, instead of a few tens. Setting up obstacle courses and running experiments would be much less arduous with such a mini robot, which could be picked up and easily carried from place to place. On the other side of the coin, building some of the components, particularly the onboard transducers and sensors, will require more delicate work than would be required on a bigger vehicle.

A New Cart

Here's a rough description of such a second generation cart. On a platform about 40 centimeters square, made mobile by two slow moving motorised wheels and a freewheeling caster, each equipped with shaft encoders, we have small sealed lead-acid cells with a capacity of about 10 amp hours at 24 volts. By the side or above these are D.C. to D.C. power converters which produce up to five amps of regulated five volts, and a few amps of regulated +24 and about an amp of + and - 12 volts. All power used by the vehicle is drawn from these supplies, insulating it from voltage fluctuations. The overall height of the vehicle is under a meter, with most of the weight being in the batteries and drive motors near the base.

A control board carries a CMOS microprocessor, with interfaces for the shaft encoders and other sensors. It has spare digital and analog channels for sensors that might possibly be added later. It talks to its fixed large computer through UARTs and high power pulsed infrared LED's and photodiodes; this greatly reduces the problems of electrical interference and works well within the confines of a single room [S1]. The effect on optical proximity detectors remains to be assessed. This problem too may be evaded by using an ultrasonic ranger. Polaroid offers the very effective one available with their cameras in an inexpensive kit.

The video system consists of a solid state camera mounted on a three degree of freedom pan/tilt/slide mechanism, precisely controlled by the microprocessor, powered perhaps by stepping motors. The picture from the camera is broadcast with a power of about half a watt in the UHF band to a nearby receiver connected to the mainframe computer through a digitizer.

Communication between the microprocessor and the mainframe is through an arpanet-like error corrected packet protocol further secured by checksumming and retransmission of packets in the absence of acknowledgements. Rod Brooks has done some work on this problem [B3].

Strength of Mind

As important as the physical vehicle, is the large processor that does its thinking. Even the level of performance of my current system taxes the DEC KL10 it runs on. The program is so slow that improvements in performance gotten by reducing the speed are just about out of the question.

Not only is the processing power taxed, but the memory limits are as well. Many of the components of the program work best when considerable amounts of data are directly accessible, from several pictures and subwindows in the vision part of the code, to distance matrices and arc lengths in the path planning. A widely available large address space machine like the DEC VAX solves the space problem (for a while), but does little (or nothing) about the speed.

About one third of the runtime in my program is devoted to extracting good pictures from a rather poor digitising system. A modern digitizer such as can be provided with new Grinnel display systems, makes the process almost instantaneous.

Perhaps half of the remaining time is spent doing array type operations such as convolutions, array summing and peak detections. A fast array processor which can be interfaced to a VAX could obviate most of that.

Some of the remaining time can be eliminated by using the address space

to good effect, simple coding efficiency improvements would also give a factor two or more in some of the later code (which I've spent less time optimising).

Combined, these techniques might speed things up by a factor of five, which would give considerable breathing room for new experiments and extensions. Farther in the future faster VAXes and multiprocessors could do arbitrarily much more.

What Then?

How does one use an improved cart connected to an improved computer? Experimenting in the effect on performance of various alternative approaches and algorithms is important, if unspectacular, work. A newly created version of the obstacle avoider would surely deserve some of that.

There are many subtle improvements possible. Determining the position of the ground by observing the 3D path of the cart is a relatively easy extension not in the current program. The long term navigational accuracy of the system is poor, depending as it does on a chain of only approximately correct coordinate transforms deduced from the vision. Some sort of long term landmark identification would be a great improvement. A by-product of such improvements would be fairly good maps of the vicinity of territory traversed.

The cart now sees by tracking patches of image from one picture to another taken from a different position. It has no means of inferring the existence of featureless obstructions such as clean walls (and the interior of smooth cardboard trees and rocks!). Others have attempted to build systems that infer the existence of extended objects from a coarse sampling of their surfaces, or from their edges.

Presently the "blocks world" systems are too restrictive, and those that work in more nearly the "real world" are too slow. Improving them, or developing better substitutes, is a worthy project. Requiring them run a real robot is an excellent way to keep the work honest.

Other "fun" projects involve programming the robot to notice holes, trenches and cliffs (inevitable disaster can be avoided by covering such hazards with glass). Much harder is a learning mode where the system could be trained to recognise objects of particular kinds, like rocks. The training might consist simply of a description given in a specially designed recognition language. Such a capability would be a great asset in a semi-autonomous planetary explorer, which would stop and ask for assistance from afar only when it encountered something new.

Such extensions can be put into the peculiar evolutionary context in which this work has been cast. After the low level vision processes like stereo ranging are working sufficiently well, (corresponding perhaps to the vertebrate optical system up to the optic chiasm), another set of procedures which integrates their outputs and produces the concept of whole objects is developed. An object classifier is subsequently built upon that, doing work which in vertebrates is probably done respectably deep in the visual cortex.

CHAPTER 12

Connections

The goal of this research was a program that could drive the cart through realistic environments. I perceived the task to be barely possible, and approached it in an uninhibited frame of mind; no code was too dirty nor method too specialized, so long as it worked. Though I expected to achieve some scientific insights, they would come as byproducts of honest work, not from a direct assault.

Stereo Vision

Computer vision, especially for rovers, is in such a tender and undeveloped state (and so marginally feasible) that very little of the prior progress in the field was of any help in writing my program. Nevertheless, my efforts can be seen to have some relation to the other work. With further development, some of the other methods might even be useful in future cart-like systems.

For understandable reasons, much of the work on extracting three dimensional information from pictures has been done with picture pairs, in imitation of human binocular vision.

Viking Mars Lander picture pairs processed by Gennery [G1] originated from a camera pair. The Mariner 9, and Apollo 15 photograph pairs processed by

Quam and Hannah [QH1], were taken by the same camera at different locations while the spacecraft were orbiting Mars and the Moon. The aerial photograph pairs used by Arnold [A1] were taken successively by a single camera aboard an airplane flying over the scene

There have been a few efforts besides mine that use more than two pictures. Baker [B4] followed edges between multiple views of an object that rotated in front of a camera, and Baumgart [B5] used the silhouettes in images obtained similarly to build models of objects, except for concavities. Burr and Chien [BC1] use picture triplets. Nevatia [N1] suggested using multiple images, taken successively by a single camera. He forsees using his techniques on orbital, planetary surface, or industrial data. In the latter case, successive images might be available of some mechanical part travelling along a conveyor belt.

Area correlation of small patches, used at times by Quam, Hannah, Nevatia, Gennery and myself is not the only way to find corresponding parts of two images.

Arnold [A1] matches edges, obtained from each image using a standard edge operator as do Burr and Chien. Quam and Hannah [QH1] match small areas, then grow them into larger areas, to obtain matches between whole shapes in the images. Marr and Poggio [MP1] use a method which examines the whole image and chooses the features matched in parallel with doing the matching.

If a sampling of depths in the image, rather than a dense depth map, is sufficient for an application, then one must select the points to range. A good idea is to pick points in one image which will be easy to find in a second. Pingle and Thomas [PT1] chose windows with high brightness variance, which also got a positive response from a corner finder.

My interest operator is a cheaper version of the same idea, though by depending on local maxima of an "interestingness" measure, it is more tolerant of regions that have low contrast. Yakimovsky and Cunningham [YC1] use an autocorrelation technique that is almost equivalent to my interest measure.

Gennery

Although the current program works quite differently, my initial approach to the obstacle avoider was with a vehicle carrying a single fixed camera. The stereo baseline in that case would be generated by the vehicle motion. Since the motion was known very imprecisely, it would have to be deduced simultaneously with the feature distances. Gennery had written a camera-solving subroutine which tackled this problem. (Fortunately the whole approach proved too error-prone. If it had not, my program would have had a major component that was someone else's prior work (shudder). It may still be used in future systems).

Gennery's camera solver [G1] is able to determine five of the six parameters which relate the relative position and orientation of two cameras viewing the same scene, from the image plane co-ordinates of five or more features points seen by both cameras. The algorithm determines the azimuth and elevation of the second camera relative to the position of the first, and the pan, tilt and roll of the second relative to the orientation of the first. The sixth parameter, the distance between the cameras, can not be determined, in principle, from image co-ordinates alone.

Gennery has developed another program that may be useful to future carts. This one deduces the position and shape of the ground directly from a stereo pair of pictures of a littered scene [G2]. It assumes the ground surface can be approximated by a two dimensional second degree polynomial (though a first

degree fit can be forced a *priori* if desired). The algorithm (which begins by finding a rough camera model using points found with my interest and correlation operators!) works with a large number of three dimensional positions of points in the scene, found with a high resolution correlator that gives probability ratings to matches, based on a detailed statistical analysis of the errors expected in two noisy images. It separates the points into ground and not ground by successively doing a least squares fit, and then selecting all points within tolerance of the candidate surface or below it for the next fitting. This has the effect of finding lower and lower surfaces, with non-ground clutter above them. Additional heuristics help eliminate spurious points.

Hannah and Quam

Quam and Hannah [QH1] combined techniques they had developed earlier, (Quam [Q1] and Hannah [H1]), to produce contour maps of portions of the Lunar and Martian surfaces from stereo pairs of photographs from the Mariner 9 and Apollo 15 spacecraft. They start their program by hand by indicating a particular planetary surface point in both images. Around this seed, the program grows a region of point pairings whose surrounding windows show a local maximum at the same disparity. When all such contiguous points have been exhausted, a nearby non-matching point is taken, and a local search performed to find a corresponding point which maximizes the correlation of the surrounding windows. If the correlation is sufficiently high, the region growing procedure is repeated. If the program fails to find any suitable new seeds, it appeals to the human operator again. Finally the depth estimates are refined by interpolating the correlation measure for a three by three cell around each point.

An early version of Gennery's camera solver then generates 3D co-ordinates of the surface points, which are used to make contour maps that sometimes look right.

Arnold

Area correlation techniques tend to fail on object boundaries because the edge of an object appears against a different background in each picture. Arnold's stereo vision system [A1] avoids this problem by matching edgels (short edge segments), extracted from the images with the Hueckel operator. He incorporates local context information in the matching process, and eventually determines a depth map of height above the ground, for the edges of the regions.

Edge detectors are unable to operate well in the presence of texture or smooth shading, so Arnold suggests that eventually edge based methods, and area correlation methods should be combined. I say the same thing.

Arnold's system was developed on aerial photographs of airport and parking lot scenes. After finding a camera model and a ground plane with Gennery's solver, the program applies a 3.19 pixel radius circular Hueckel operator to both images, typically producing around 1200 edgels. Each edge has position, angle, and the brightness on each side. The spatial information is normalized using the information from the camera solver and ground finder, starkly limiting the search area for matching edgels. Thresholds on the allowable differences between angles and brightness levels further constrain the plausible matches. Typically there might be 8 possible matches in the right hand image for an edgel in the left. Disparities are calculated for each of these possible matches and attached to the edges in one of the pictures.

Local context further resolves the ambiguity. Two edgels are considered to belong to the same physical edge if they are within 3 pixels and 90° of each other, and the angle of a line connecting their centers lies between their angles and the brightness levels are consistent on at least one side. A voting scheme weighted by edgel distances from each other and by disparity disagreements finds a likely disparity, and hence a height above the ground, for such physical edges.

Burr and Chien

Burr's and Chien's system [BC1] is given piece-wise linear wire frame models of objects to be found in scenes. It assumes three fixed cameras, one in the center, another rotated 20° east, and a third rotated 10° north. The program compares center and east pictures when matching vertical edges, and center and north for horizontal edges. Depth maps are made using the region matching algorithms of Hannah [H1], then an edge operator is used on the center image. Edgels are linked into chains of near neighbors with consistent directionality. Then the depth maps guide a local search in one of the two other images for a matching edge. The 3D edges so obtained are matched against the wire frame models to locate objects in the scene.

Mori et al.

Mori et al. [MKA1] try to make extremely accurate depth maps from aerial photographs. The camera model is determined from accurately known ground points. A fixed size correlation window is inadequate for the desired accuracy because a patch of sloped surface can vary significantly in size between two views.

They make an initial approximation using a standard feature correlator, then correct one of the images for distortions that the estimated ground surface gradients would have caused. The process is iterated, and eventually an accurate map is produced.

Yakimovsky and Cunningham

Yakimovsky's and Cunningham's [YC1] stereo program was developed for the JPL Robotics Research Vehicle, a testbed for a Mars rover and remote processing systems (Levine et al. [LOY1] describe an earlier stereo system using the same hardware). The RRV has a pair of rigidly attached cameras, with highly linear lenses, so the search space is a very narrow line parallel to the projection of the stereo axis. A small correlation window is chosen in one image and expanded until its correlation with itself shifted one pixel is sufficiently poor, showing it has adequate information content, or until it gets too large (this is essentially an interest operator step). Windows that pass this test are correlated with the other image.

Marr and Poggio

Marr and Poggio [MP1] have a different approach. Their matching pattern has no precise edges, it is fuzzy and implicit from a cooperative (or relaxation) algorithm in which independent measurements made at nearby parts of the image inhibit or reinforce each other. They start with two general principles; Each image point may be assigned at most one disparity and, except for a few boundaries, disparity varies smoothly almost everywhere.

For each pixel in one image and for a range of disparities there is a node which represents the likelihood that the corresponding pixel in the other image represents the same physical object. The nodes are initialised by a simple pixel comparison. An iterative process then repeatedly updates each pixel/disparity node with a normalised measure of how surrounding nodes of the same disparity agree.

Marr's and Poggio's program works well on random-dot stereograms. (Juless [J1]), taking about 15 iterations to produce a solid representation of the hidden figures.

In 1979 Marr and Poggio [MP2] published and Grimson and Marr [GM1] implemented a different approach that involves low pass filtering the images to various resolutions, then correlating on the zero crossings in the filtered images. A low frequency version provides a coarse positioning, and the less filtered images are then used to refine the positions. In some ways the process is similar to the binary search correlation method described in this thesis.

Path Planning

Plausible or minimum cost path finding is a simpler, and in an abstract sense, less interesting problem than stereo vision. A number of approaches have been published, some heuristic, many depending on exhaustive search. Some of these depend intimately on a "Manhattan distance"(runs exactly in X or Y directions exclusively) geometry, which are good for circuit board layout, but uninteresting to us.

Many of the more general approaches, including the one implemented in

the cart program, translate the problem into a connected graph with arc costs. The best general solution for finding the shortest distance in such a graph is still the $O(n^2)$ algorithm published by Dijkstra [D1]. My program uses a variant of it.

A number of heuristic methods have been presented. SRI's Shakey system used one such. A generalization is described by Hart, Nilsson and Raphael [HNR1]. They call it the A* algorithm.

Recently Lozano-Pérez and Wesley [LW1] published an obstacle avoidance algorithm whose core is nearly identical to mine. Instead of circular obstacles, theirs avoids polygons. Instead of circle to circle tangents, they consider vertex to vertex paths (which is a little easier, because there are fewer vertices than tangents). Their program, like mine, maps the possible path space into a graph, and solves with an efficient graph algorithm.

Appendix 1

Introduction

Software Pointers

It has not been possible in this thesis to present all the details of all algorithms which made the cart go. Further information might be gleaned from the code itself. The Stanford AI lab "Dart" archiving system may make the relevant computer files available a number of years after the 1980 publication of this document. Though changes in the configuration of the system will rapidly make many of the programs inoperable, some portions may be salvageable. The names of involved files follow.

SPOTS.SAI[VIM,HPM] is the calibration program. It must be told the name of a camera or a file containing a picture of the spot array seen by the camera, and how far (in inter-spot spacings) the lens center of the camera was from the array. The camera orientation is assumed to be horizontal (by definition), and lens center the same height as the central spot of the array. The program writes out a calibration file.

CRUSH.SAI[MIS,HPM] is the obstacle avoider. It asks for a list of TV receivers, the tilt of the camera from the horizontal (it was 15° in all my runs)

and several other reasonably self evident questions.

CRASHS.SUB[MIS,HPM] is a file of Sail macros used by CRUSH.SAI, which contains most of the actual substance of the program.

CRUSH.SAI also uses a set of externally compiled procedures referenced through the files VIGHDR.SAI[VIS,HPM] (display and image handling utilities), REDPIC.SAI[VIS,HPM] (which facilitates handling sequences of reduced pictures, and also feature description window sequences), DISTOR.HDR[VIS,HPM] (written by Donald Gennery, which derive and apply least-squares camera distortion correcting polynomials), and RD1HOG.SAI[CAR,HPM] (which contains the two-arc atomic path planner, the cart response simulator and the remote control code).

VIGHDR.SAI itself calls in the files PIXHDR.SAI[VIS,HPM] (picture handling utilities; outlined in Appendix 10) and GRAHDR.SAI[GOD,HPM] (the GOD device-independent graphics package, also summarized in Appendix 10). It also calls in VIXFAI.REL[VIS,HPM] (which is a package for displaying pictures in the environment of the older data-disc display package. It is compiled from the FAIL source file VIXFAI.FAI[VIS,HPM]), for one of its minor routines.

PIXHDR.SAI calls in PIXSAI.REL[VIS,HPM] (compiled from PIXSAI.SAI), SQTILE.REL[VIS,HPM] (compiled from SQTILE.SAI) and PIXFAI.REL[VIS,HPM] (compiled from PIXFAI.FAI).

GRAHDR.SAI invokes GRASAI.REL[GOD,HPM] (from GRASAI.SAI) and FILHDR.SAI[VIS,HPM] (an extended file name parser which itself calls FILNAM.REL[VIS,HPM] compiled from FILNAM.SAI).

AVOID.SAI[CAR,HPM] and AVOID1.SAI[CAR,HPM] are demonstration

programs for the approximate and exact, respectively, path planning algorithms. They are presented in full in Appendix 8.

NCOARP.SAI[CAR,HPM] demonstrates the interest operator and binary search correlator.

CALTST.SAI[MIS,HPM] produces polynomial-overlayed calibration test patterns, of which Figure 4-4 is an instance.

CART.SAI[TRI,HPM] (using 3DRAW.SAI[TRI,HPM]) generates files which can be used to generate the 3D line drawings and gray scale graphics like the illustrations in Chapter 9. VIEWDD.SAI, VIEWXG.SAI, VIEWMS.SAI, SEEGOD.SAI take these files and produce representations on data discs, the XGP, picture files and GOD files, respectively.

TYPHDR.SAI[GOD,HPM] is an extension of the GOD file routines which allows convenient typesetting of two dimensional text in mixed fonts.

CIRCUT.SAI[GOD,HPM] is another little extension that facilitates drawing of logic and circuit diagrams, such as the one in Figure 10-2. The diagram was actually produced by BATCH2.SAI[DIA,HPM].

Hardware Overview

The cart hardware on which the obstacle runs were based is of even more transient interest than the software. A brief overview is presented here.

The control system is very simple and inaccurate. Packets of six bits are passed from the KL-10 output device known as the CAR, through the Cart

Control Interface attached to the PDP10, over a unidirectional CB radio transmitter. These six bits are given the following interpretations: Drive motor on; Reverse/forward; Steer left; Steer right; Slide camera left; Slide camera right.

If both bits are on for steering a centering process takes place, using potentiometers attached to the relevant device and a simple op-amp network. To avoid spurious noise-induced camera motion, the slider is activated only by the protocol: both slider bits off; both bits on; bit for appropriate direction on. After receiving such a sequence the camera slides a number of stepping motor increments indicated by a thumbwheel setting on its chassis; typically 256 steps.

There are only two states for the motors - going or not going. No smooth power control is possible. Because of the current data transmission scheme it is only possible to specify the duration of any motor actuation to an accuracy of at best one thirtieth of second. Thus in general after issuing any control command it is impossible to determine whether the desired action has been carried out. Even assuming that the motors have run for the desired time period the positions of any of the actuators, and the cart itself, are uncertain because of the rough motions caused by switching full power on and off to the motors.

The KL-10 device CAR can be used to control the on-off state of the power transistors labelled 20 through 26 in the cart control interface. Device CAR is activated every tick (sixtieth of a second) and the state of these transistors is updated if indicated by the output buffer. When running, CAR also keeps transistor 18 off. Alternatively CONO's can be used to control transistors 18 through 35 directly, however accurate timing will be lost even if a SPACEWAR module is used.

Transistor 18 operates a relay which when off causes the cart transmitter to

be on. The transmitter encoder has its own clock (running at about 1kH). It scans transistors 21 through 26, emitting a short (order of 0.5 mS) pulse before looking at each one, and then idles for either two or four cycles depending on whether the transistor currently under examination is on or off. After looking at the sixth transistor and idling for the appropriate amount of time it emits another pulse and then idles for a longer period before starting the scan again. Thus the six bits are encoded as either short (= 1) or long (= 0) gaps between seven high pulses, with a gap roughly as long as each pulse train between successive transmissions of the train.

The receiver circuitry on the cart passes on this train of seven positive going pulses to the onboard logic. The onboard logic has three six bit registers, call them A, B, and C. Register A is a shift register which receives 0's and 1's decoded from the intergap pulses by some timing circuitry. When a sufficient time has passed, since the reception of a pulse, and if the number of pulses was exactly six, bits in A are taken to be a packet. A single 10 micro-second control pulse is emitted from the timing network. On the leading edge of this pulse, registers A and B are compared. If their contents are identical the contents of B are shifted in parallel to C and the outputs of C, which pass through a static network to drive the relays operating the motors, are enabled by a retriggerable one shot for approximately two seconds. Regardless of the result of the comparison the contents of register A are shifted in parallel into register B on the trailing edge of the control pulse.

Thus the logic compares successive sets of six pulses and when they are identical uses them to control the actuators. Although the retriggerable one shot mentioned above enables the control register C for approximately two seconds, a new pair of successive identical bit packets will overwrite the contents of that

register and retrigger the one shot for another two seconds. Thus to terminate some motor activity packets of six zeroes can be sent repeatedly.

In case there is any future interest in this dull hardware, I have left a set of circuit diagrams with Allan Miller of the Stanford AI lab.

History

The following chronology, which goes to 1974, was written in 1974 just before the meeting that resulted in the acquisition of the new cart TV transmitter and my first foray into cart vision. It provides a historical perspective on the history of Chapter 2. Many of the emotions expressed in it were fleeting.

1966

A cart, thrown together by Mechanical Engineering for a short term research project, becomes available to the AI Lab. Les Earnest seizes on this as an opportunity to add mobility to the lab's bag of tricks, and convinces Rod Schmidt to do his thesis on the subject.

1967-1971

Schmidt spends several years single-handedly constructing radio links between the cart and the PDP 6. In this he values quick completion over high quality, and succeeds at least in minimizing the quality. It quickly becomes obvious that the model airplane control link (unlike the link used by M.E., presumably) is incapable of specifying the position of the four wheel steering (which has a range of

two full turns) with manageable accuracy (a 5% jitter in the control pulse width is a 30° change in the wheel orientation). His solution is conversion of the steering to a conventional arrangement, which requires a much smaller total range of steering angles. Vic Scheinman helps, but his emphasis too is on speed over sturdiness. The first few attempts don't work very well, because simple connection of the front wheels with a chain drive causes them to want to go in slightly different directions when pointed anywhere but straight ahead. A bar linkage is finally installed. This doesn't work very well either, because of the short wheel base, but everyone is tired, and in any case there are no other new ideas. The result is a general mechanical degradation. Schmidt, now under the guidance of McCarthy, uses this vehicle to write a thesis concerning automatic cars. Realising how long he has already been here, he applies to the thesis the same attitude that has made the hardware what it is, and completes it with the minimum possible expenditure of effort. He is aware of the state of things, and includes in it (on p 151) the paragraph:

"In conclusion let me comment that this vehicle was intended as a low-budget way of getting some experience with driving problems, and that if this research is continued, a new vehicle should be obtained. The limitations on operating time caused by battery discharge and the lack of equipment space, nonexistent suspension, and low speed of the present vehicle will make future work increasingly difficult and unrealistic."

The silence is deafening.

1970-1973

Baumgart decides he likes the idea of a robot that reasons visually, and

concocts a grand scheme in which every scene viewed by the camera would be related to a model of the lab and surrounding territory. He notices the uncertainty in the analog link, and decides make it into a digital one. This is his first digital design effort, and the result, which provides for on-off control of the motors and has no indication of the orientation of anything, is considerably inferior to the original in concept, and in addition works unreliably. The original servo electronics are disassembled or misplaced, making his changes irreversible. He rationalizes that the problems with the link are unimportant, since, when his visual reasoner works, it will be able to deduce the state of things, and detect when a transmitted command has failed, to try again. The enormity of the effort needed to make his plan a reality becomes apparent to him as he works on sub-problems. Since it would become possible to actually use a vehicle only when his proposed scheme was almost completed, and since he now sees that it is unrealistic to think that it could be brought to fruition in a reasonable number of years, he abandons any serious efforts directly concerned with the cart, but maintains his association with it, as a status symbol and a toy. He occasionally drives it around for show, often over rough ground, contributing to its mechanical decline. During this time several other graduate students are steered towards this essentially non-existent "cart project". They are disillusioned by the lack of a coherent plan and suffer from too little guidance and from conflicts with Baumgart's personality. All these associations are short lived and unhappy. Baumgart finds success and happiness working on the graphics and vision sub-problems suggested by his original concept.

1973-1974

Quam, having done his thesis on the processing of Mars pictures, expresses an interest in using the cart for Mars rover research. He has picture processing algorithms, but little else. Interfacing with Baumgart is impractical, so nothing much happens. Moravec, who had decided he liked the idea of an automatic car before he came here, and who had wanted to come to Stanford largely because of that, decides it is time to start serious work towards a thesis on this topic. He is steered towards Quam, who, at this stage, has little to offer other than admonitions to "get this show on the road". He trys, rebuilds the control link to make it a little more reasonable, and plays with the vehicle for a few weeks, contributing to further deterioration in its already delicate mechanical condition. Then, in collaboration with Quam, who is tuning up his picture correlation programs, he writes a control package for vehicle. Before these two efforts can be combined, he throws a wrench into the works by running the cart off a ramp, which provides the excuse to stop working that the TV transmitter had been looking for from the day it was born. He spends six months trying to recover from the effects of his folly, with only limited success. Very frustrated and anxious to get back to productive work, he writes a proposal to McCarthy and Earnest, begging for help out of his predicament. For similar reasons, on a slightly larger scale, Quam writes a proposal to his friends at JPL.

The story to this point was an example of that well known conservation law "you can't get something for nothing". By definition, there can hardly be a cart project without a cart, since in the absence of a vehicle we have only computer vision. The impression grows on me that what we have now is worse than no vehicle at all. At no time has the state of the cart hardware ever approached a

level comparable to that of the hand-eye in the Rancho arm days, yet the comparison between the two projects is undoubtedly behind the hostility the cart now faces. I wonder if any arm theses would have gotten beyond the first few months if the arm required massive repairs after every half hour of operation, and if these repairs had to be effected by the student trying to write the programs. This situation is exactly the one that has faced everyone considering doing a cart thesis. It instantly excludes those who are not hardware inclined, and makes technicians of those who are.

Overview

The following eleven pages contain pictures of some of the obstacle avoider's internal state from step to step in a successful run.

Each diagram is a plan view of the world at a stopping position. The grid cells are two meter squares, conceptually on the floor.

The cart's own position is indicated by the small square that moves from diagram to diagram. The third component of its position, namely its height, is shown by the graph, calibrated in centimeters, to the left of grid. Since the cart never actually leaves or penetrates the floor, this graph provides an indication of the overall accuracy of the system.

The irregular, tick marked, line behind the cart's position is the past itinerary of the cart (as deduced by the program). Each tick mark represents a stopping place.

The picture at top of the diagrams is the view seen by the TV camera at the various positions. The two rays projecting forward from the cart position show the horizontal boundaries of the camera's field of view (as deduced by the camera calibration program).

The numbered circles in the plan view are features located and tracked by

the program. The centers of the circles are the vertical projections of the feature positions onto the ground. The size of each circle is the uncertainty (caused by finite camera resolution) in the feature's position. The length of the 45° line projecting to the upper right, and terminated by an identifying number, is the height of the feature above the ground, to the same scale as the floor grid.

The features are also marked in the camera view, in the guise of numbered boxes. The thin line projecting from each box to a lower blob is a stalk which just reaches the ground, in the spirit of the 45° lines in the plan view.

The irregular line radiating forwards from the cart is the planned future path. This changes from stop to stop, as the cart fails to obey instructions properly, and as new obstacles are detected. The small ellipse a short distance ahead of the cart along the planned path is the planned position of the next stop.

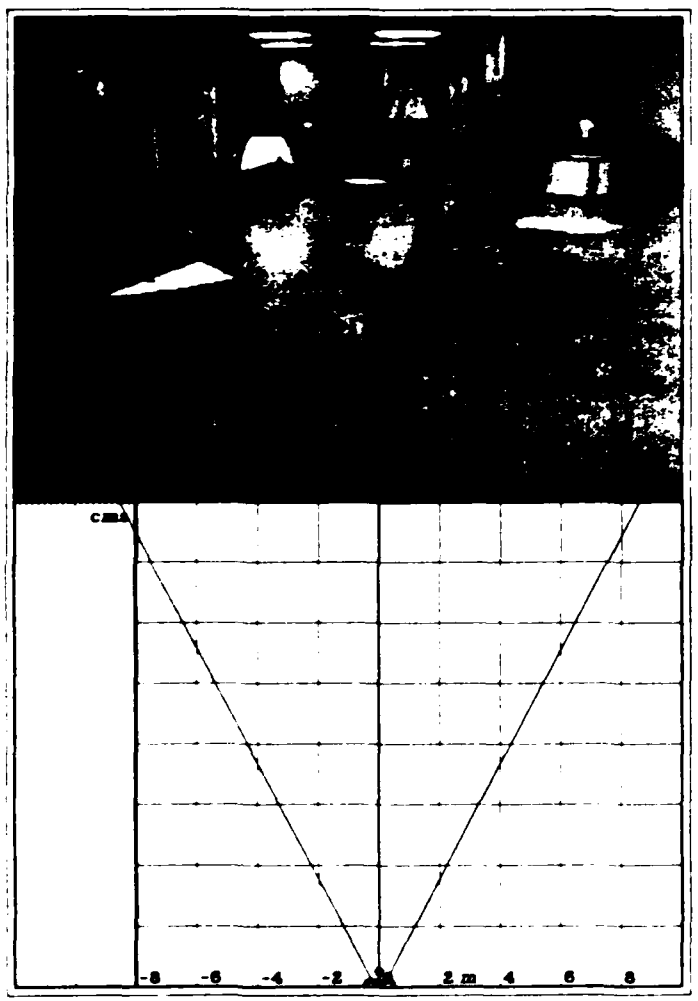


Figure A3-1: *The initial position Requested destination is 16 meters ahead*

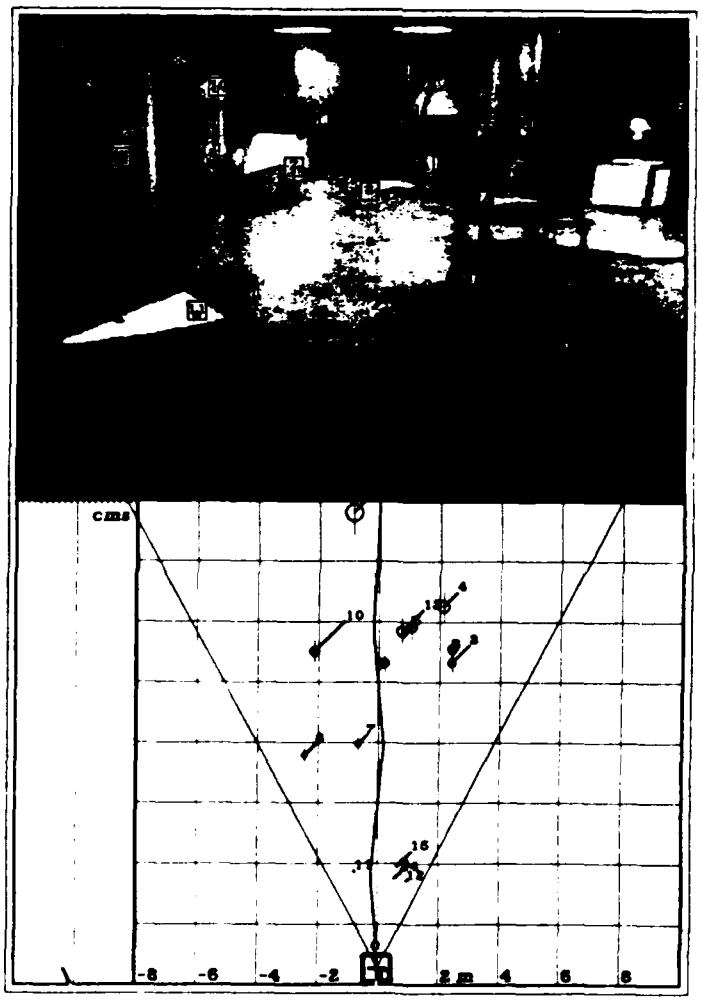


Figure A3-2: *The second stop. Three obstacles to avoid.*

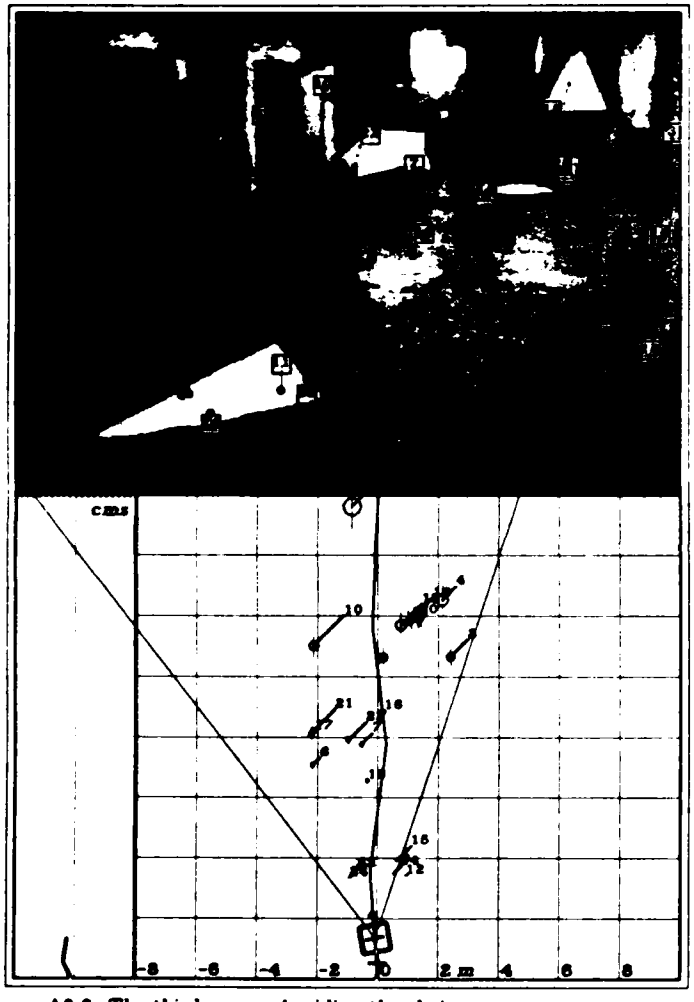


Figure A3-3: *The third pause. Avoiding the chair.*

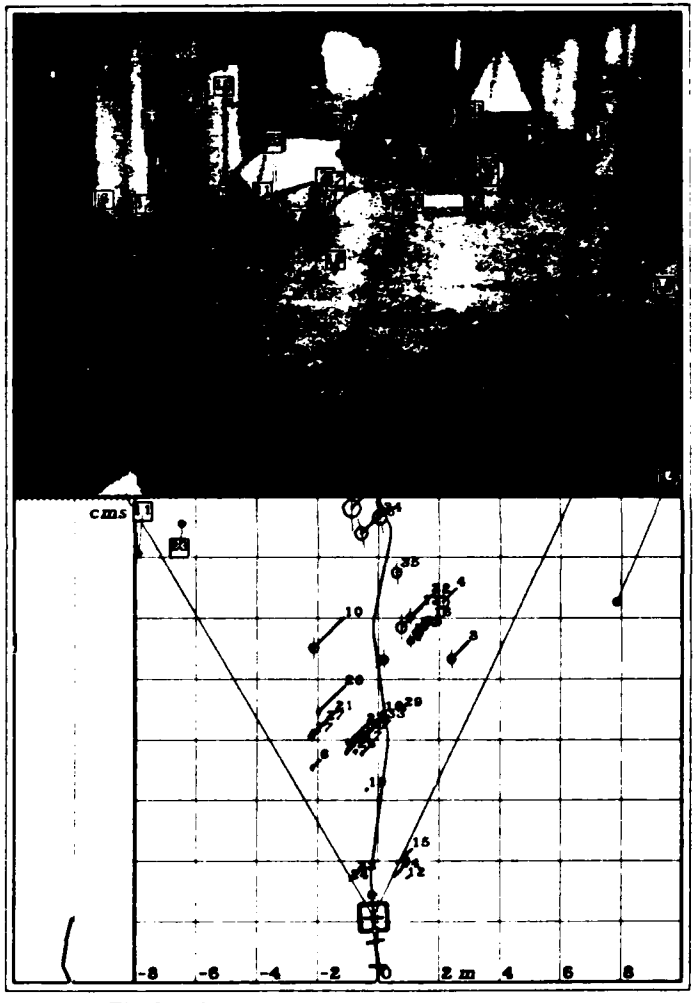


Figure A3-4: *The fourth stop.*

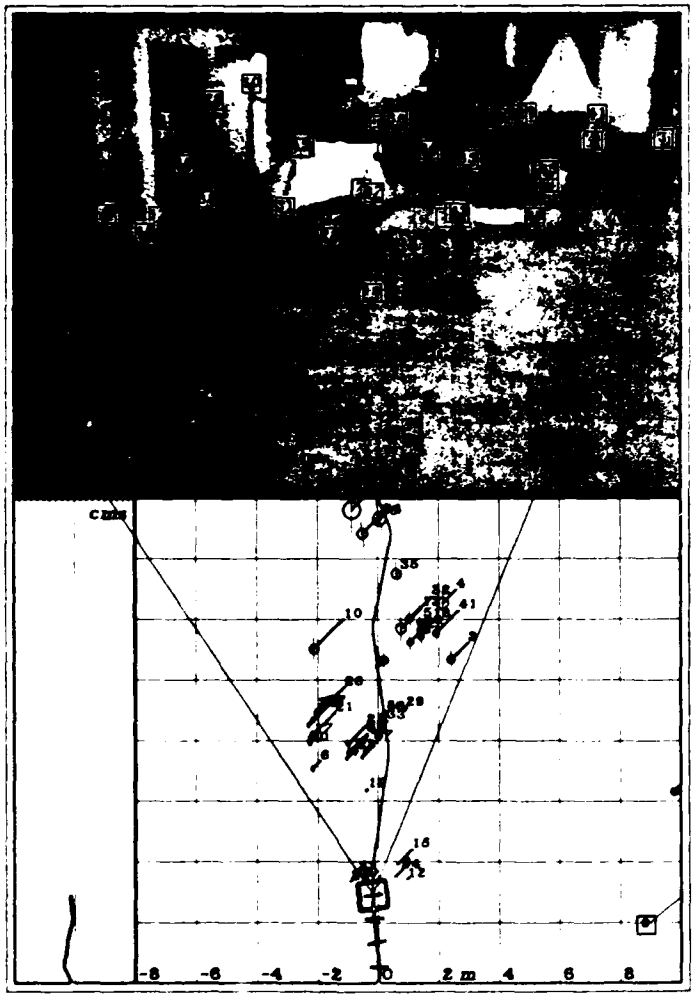


Figure A3-5: *The fifth stop.*

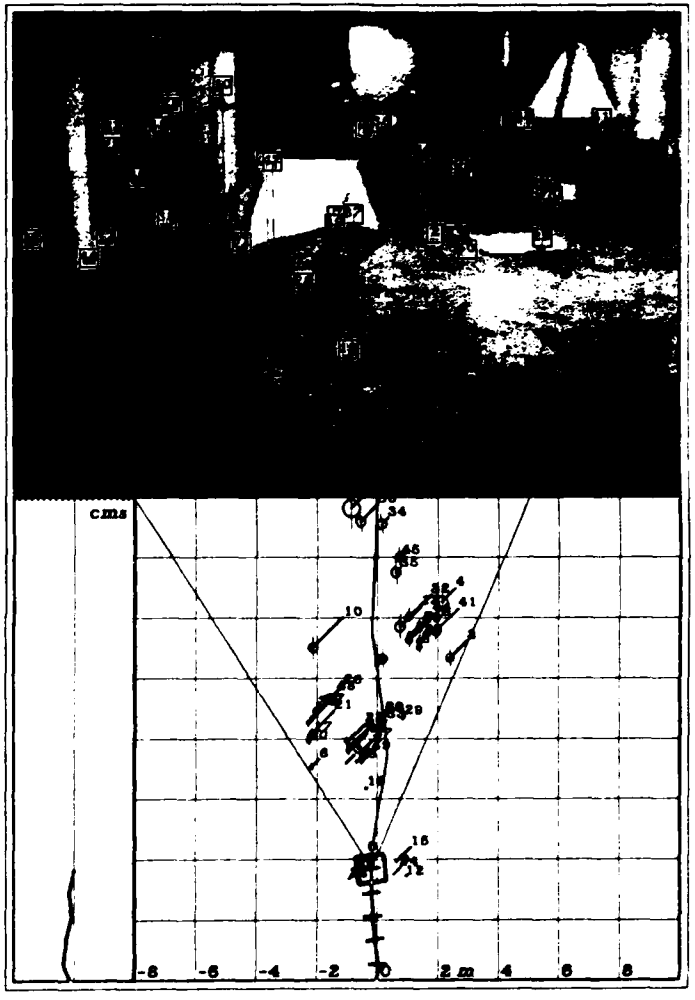


Figure A3-6: *The sixth stop.*

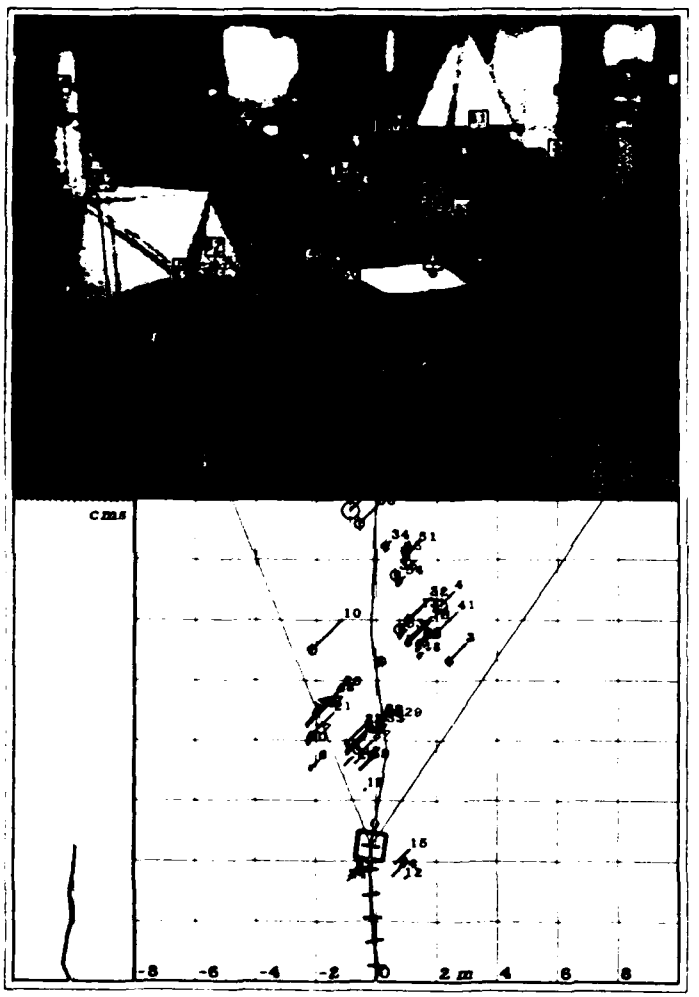


Figure A3-7: *The seventh stop. Now to get around the icosahedron.*

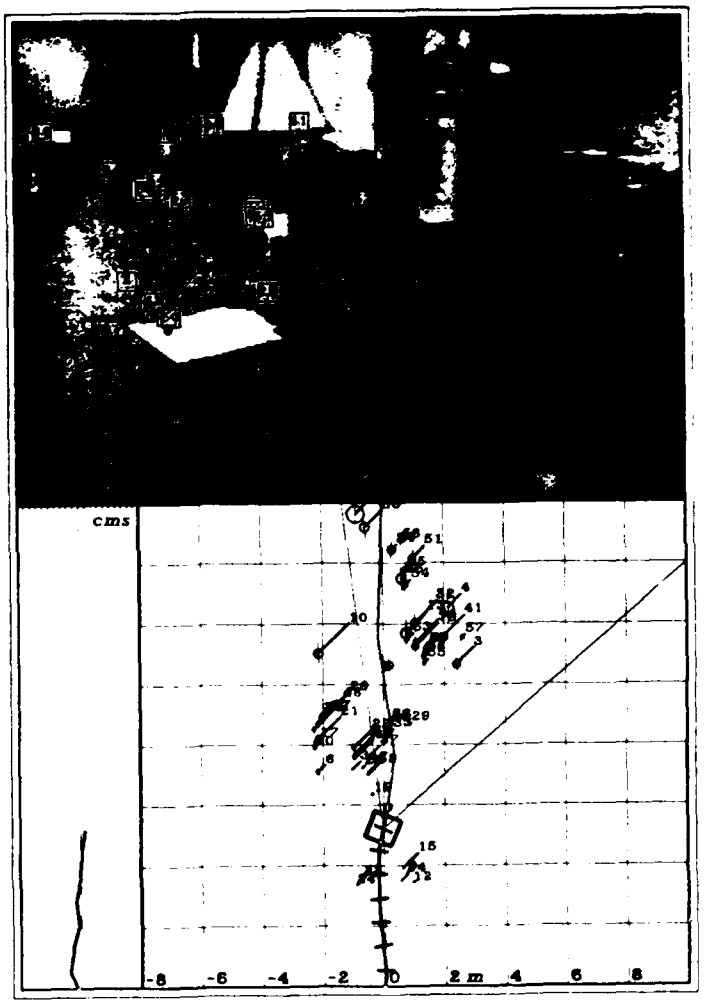


Figure A3-8: *The eighth stop.*

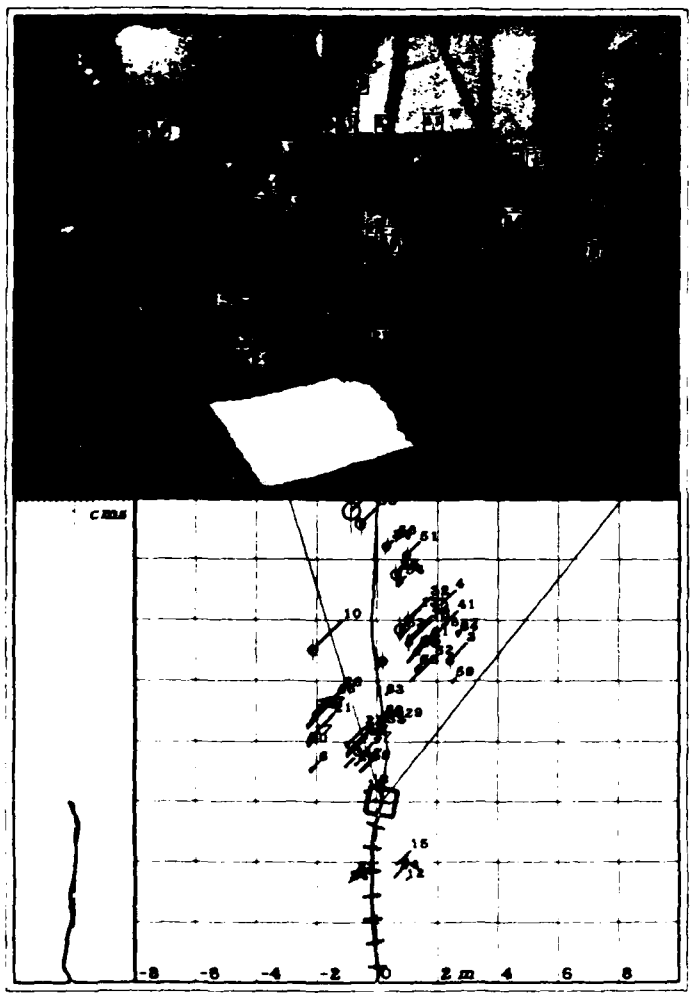


Figure A3-9: *The ninth stop The world model is drifting.*

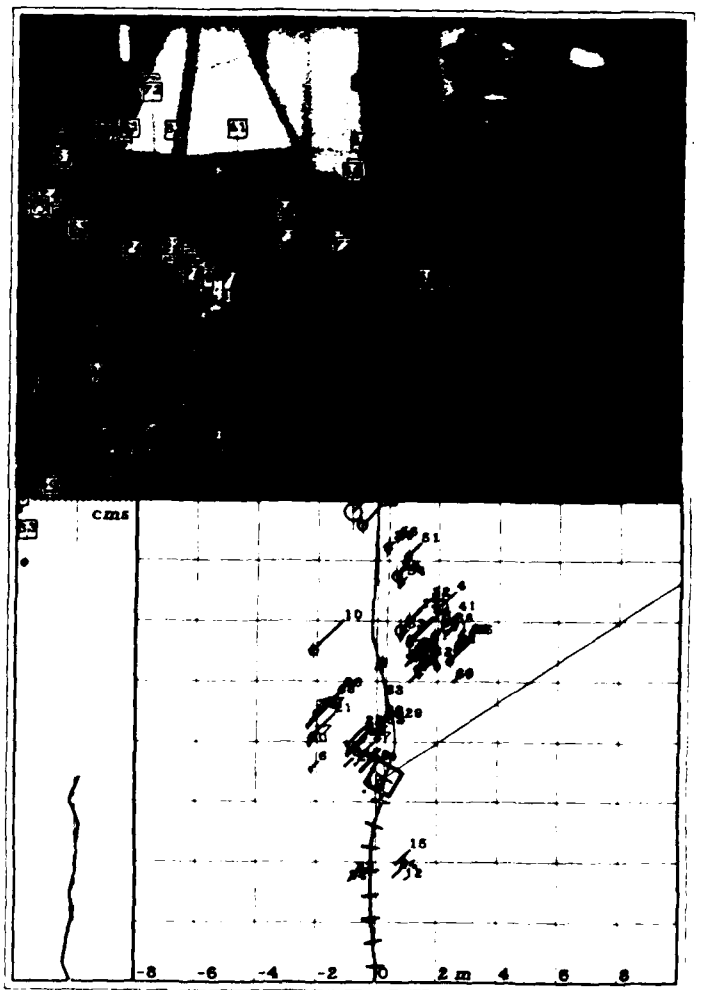


Figure A3-10: *The tenth stop.*

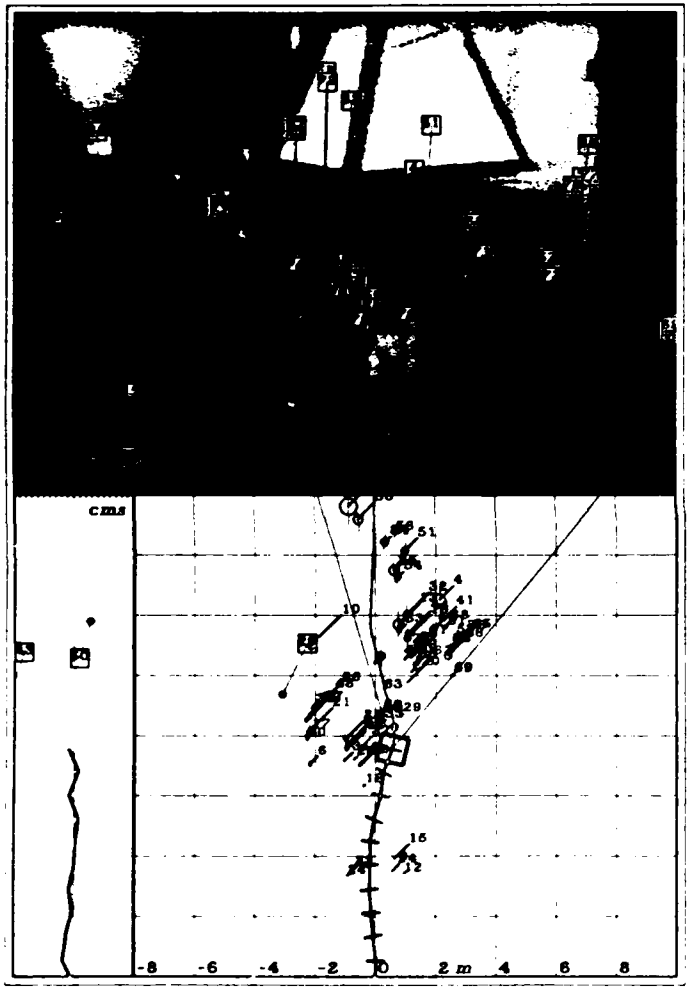


Figure A3-11: *The eleventh stop. Time to skirt the fake tree*

Correlation

Correlation coefficients

The problem is, given two n by n arrays of pixels, referred to as $A_{i,j}$ and $B_{i,j}$, determine how alike they are.

One of the simplest comparisons is

$$\sum_{i,j=0,0}^{n,n} (A_{i,j} - B_{i,j})^2$$

If the camera did exact photometric measurements, and if lighting conditions remained unchanged between frames, this measure would be ideal. Unfortunately the camera response varies from place to place on the camera field due to the nature of the optics and complicated vidicon effects, from scene to scene because of target voltage response to illumination, and from time to time due to changes in battery voltage. In spite of these drawbacks the measure has been used successfully.

Normalized Correlation

Another alternative is "normalized correlation", the root mean square

average displacement of co-ordinate pairs $(A_{i,j}, B_{i,j})$ from a least squares regression line relating A to B. Let $a_{i,j} = A_{i,j} - \overline{A}$ and $b_{i,j} = B_{i,j} - \overline{B}$, $\overline{A}$ and $\overline{B}$ being the means over the entire window. The normalised correlation coefficient is defined as:

$$\sigma = \frac{\sum a_{i,j} b_{i,j}}{\sqrt{\sum a_{i,j}^2 \sum b_{i,j}^2}}$$

Normalised correlation is invariant under all linear transformations of the values of A and B. The complete contrast insensitivity this implies is a needless waste of information, because contrast changes in actual pictures are relatively small from frame to frame. In addition, the measure has a degeneracy when all the values of either A or B are the same. This will happen often if the search window contains uniformly shaded areas, and must be handled by a special test. A different measure is called for.

Pseudo-normalized Correlation

Although the correlation coefficient itself remains unchanged, the "line of best fit" is different when calculated from A to B than from B to A. For the best fit of A to B, $b = ka$, we wish to find k which minimizes $\sum(ka - b)^2$. This is $k = \sum ab / \sum a^2$. The line can be expressed

$$a = \frac{\sigma\sqrt{\sum a^2}}{\sqrt{\sum b^2}} b$$

for the best fit of B to A, $a = kb$, we wish to find k which minimizes $\sum(kb - a)^2$. This is $k = \sum ab / \sum b^2$. The line can be expressed

$$b = \frac{\sigma\sqrt{\sum b^2}}{\sqrt{\sum a^2}} a$$

these are equivalent if σ is unity, i.e. if the correlation is perfect. Since for our application, the comparison of two picture patches, there is no particular reason for choosing one over the other, we compromise and choose a line equidistant from the two, removing the asymmetry between A and B:

$$b = \frac{\sqrt{\sum b^2}}{\sqrt{\sum a^2}} a$$

or equivalently

$$a = \frac{\sqrt{\sum a^2}}{\sqrt{\sum b^2}} b$$

We will use this to obtain a correlation measure *more* suited to our needs.

Small contrast changes should be tolerated, large ones should not. Contrast is expressed by the slope of the line of best fit, a slope of unity denoting a perfect match, slopes of zero and infinity indicating that one or the other of the patches has no variations in intensity. Consider the normalized dot product of the line of best fit with a line of unity slope. Since this is the cosine of the angle between them, it will be unity for a perfect match, and very near unity for small contrast differences, since the cosine is flat around zero angle, and zero for a negative correlation. If we were to multiply this dot product by the normalized correlation coefficient, we would obtain a measure which was unity only for perfect correlation with no contrast differences, dropping rapidly with lessening correlation, and slowly at first and then rapidly with increasing contrast differences.

This measure still has a few flaws. If one of the patches is without intensity variations, the normalized correlation coefficient becomes 0/0 and the dot product $1/\sqrt{2}$. In fact we want our measure to be zero in this case. This can be arranged if we multiply the normalized coefficient by the cosine of *twice* the angle between the

correlation line and a line of slope 1. This cosine will be zero for lines of slope zero or infinity, and the limit of the product with the normalised correlation coefficient will also be zero in those cases. In addition, perfect negative correlation will again appear as minus unity, which might be an advantage in some circumstances.

Deriving an expression for this measure, we note that the cosine of our "compromise" correlation line with one of unity slope is

$$\frac{\sqrt{\sum a^2}+\sqrt{\sum b^2}}{\sqrt{2(\sum a^2+\sum b^2)}}$$

To convert this to the cosine of twice the angle with $a = b$, we use the identity

$$\cos(2t) = 2\cos^2(t) - 1$$

giving us

$$\frac{2\left(\sqrt{\sum a^2}+\sqrt{\sum b^2}\right)^2}{2(\sum a^2+\sum b^2)} - 1 = \frac{2\sqrt{\sum a^2 \sum b^2}}{\sum a^2+\sum b^2}$$

multiplying by the normalized correlation coefficient, we get

$$\frac{2\sqrt{\sum a^2 \sum b^2}}{\sum a^2+\sum b^2}\frac{\sum ab}{\sqrt{\sum a^2 \sum b^2}} = \frac{2\sum ab}{\sum a^2+\sum b^2}$$

Note that whereas normalized correlation consists of the sum of the pair-wise products of A and B divided by the geometric mean of the sum of their squares, this new measure, *referred to as pseudo-normalized correlation*, is the sum of the products divided by the *arithmetic* mean of the sums of the squares. Since it involves an addition and a halving rather than a multiplication and a square root, it is actually easier to calculate. The prettiness of the result leads me to suspect that a more elegant derivation exists.

This pseudo-normalized measure works as well as any of the alternatives, and is easy to compute. It is the standard correlation measure in cart software.

Appendix 7

Stereo

Motion Stereo Mathematics

Here we go over the ground covered briefly in the **Motion Stereo** section of Chapter 7 in gory detail.

We want to find the three dimensional rotation and translation that, if applied to the co-ordinates of the features at the first position, minimises the weighted sum of the squares of the distances between the transformed first co-ordinates and the raw co-ordinates of the corresponding points at the second position.

The least squares expression to be minimized is

$$Error = \sum_{i=1}^{N} \frac{(X_{1,i}-X'_{2,i})^2+(Y_{1,i}-Y'_{2,i})^2+(Z_{1,i}-Z'_{2,i})^2}{U_i^2}$$

where the $X_{1,i}$, $Y_{1,i}$ and $Z_{1,i}$ are the co-ordinates of the features as observed from the first cart position. $X_{2,i}$, $Y_{2,i}$ and $Z_{2,i}$ are the co-ordinates of the points as seen from the second position. $X'_{2,i}$, $Y'_{2,i}$ and $Z'_{2,i}$ are the points from the second position subjected to a co-ordinate transformation in the attempt to make them match the first set. U_i is the combined uncertainty in the position of feature i

in both frames. To get the most out of our data, we need to weight features proportional to their accuracy. To simplify the subsequent formulas, the U_i's are normalised to make the sum of their inverse squares unity.

For a general linear transformation, these values are defined by

$$[X'_{2,i}, Y'_{2,i}, Z'_{2,i}] = [X_{2,i}, Y_{2,i}, Z_{2,i}] \begin{pmatrix} AA & AB & AC \\ BA & BB & BC \\ CA & CB & CC \end{pmatrix} + [X_c, Y_c, Z_c]$$

where AA through CC is the "rotation" part of the transform and X_c through Z_c is the translation.

Error can be expanded out and the summation can be distributed over the terms, giving us

$$\begin{aligned} Error = {} & X_c^2 + Y_c^2 + Z_c^2 + \overline{X_1^2} + \overline{Y_1^2} + \overline{Z_1^2} \\ & + (AA^2 + AB^2 + AC^2)\,\overline{X_2^2} \\ & + (BA^2 + BB^2 + BC^2)\,\overline{Y_2^2} \\ & + (CA^2 + CB^2 + CC^2)\,\overline{Z_2^2} \\ & - 2\,(\; AA\,\overline{X_1 X_2} + AB\,\overline{Y_1 X_2} + AC\,\overline{Z_1 X_2} \\ & + BA\,\overline{X_1 Y_2} + BB\,\overline{Y_1 Y_2} + BC\,\overline{Z_1 Y_2} \\ & + CA\,\overline{X_1 Z_2} + CB\,\overline{Y_1 Z_2} + CC\,\overline{Z_1 Z_2} \\ & - (AA\,BA + AB\,BB + AC\,BC)\,\overline{X_2 Y_2} \\ & - (AA\,CA + AB\,CB + AC\,CC)\,\overline{X_2 Z_2} \\ & - (BA\,CA + BB\,CB + BC\,CC)\,\overline{Y_2 Z_2} \\ & + (\overline{X_1} - AA\,\overline{X_2} - BA\,\overline{Y_2} - CA\,\overline{Z_2})\,X_c \\ & + (\overline{Y_1} - AB\,\overline{X_2} - BB\,\overline{Y_2} - CB\,\overline{Z_2})\,Y_c \\ & + (\overline{Z_1} - AC\,\overline{X_2} - BC\,\overline{Y_2} - CC\,\overline{Z_2})\,Z_c \;) \end{aligned}$$

where $\overline{X_1}$ is $\sum_i \frac{X_{1,i}}{U_i^2}$ and $\overline{X_1 Z_2}$ is $\sum_i \frac{X_{1,i} Z_{2,i}}{U_i^2}$ and similarly for the other barred

quantities. With the expression in this form, the barred quantities can be evaluated once and for all at the start of the solving.

Error can be differentiated partially with respect to X_c, Y_c and Z_c. If the resulting linear expressions are equated to zero and solved, we obtain

$$\begin{aligned} X_c &= \overline{X_1} - AA\,\overline{X_2} - BA\,\overline{Y_2} - CA\,\overline{Z_2} \\ Y_c &= \overline{Y_1} - AB\,\overline{X_2} - BB\,\overline{Y_2} - CB\,\overline{Z_2} \\ Z_c &= \overline{Z_1} - AC\,\overline{X_2} - BC\,\overline{Y_2} - CC\,\overline{Z_2} \end{aligned}$$

These are the components of the translation vector that minimizes *Error* for a general "rotation" matrix. They can be back substituted into *Error*, making it a function of the rotation alone.

$$\begin{aligned} Error = {} & \overline{X_1^2} + \overline{Y_1^2} + \overline{Z_1^2} \\ & + (AA^2 + AB^2 + AC^2)\,\overline{X_2^2} \\ & + (BA^2 + BB^2 + BC^2)\,\overline{Y_2^2} \\ & + (CA^2 + CB^2 + CC^2)\,\overline{Z_2^2} \\ & - 2\,(\; AA\,\overline{X_1 X_2} + AB\,\overline{Y_1 X_2} + AC\,\overline{Z_1 X_2} \\ & + BA\,\overline{X_1 Y_2} + BB\,\overline{Y_1 Y_2} + BC\,\overline{Z_1 Y_2} \\ & + CA\,\overline{X_1 Z_2} + CB\,\overline{Y_1 Z_2} + CC\,\overline{Z_1 Z_2} \\ & - (AA\,BA + AB\,BB + AC\,BC)\,\overline{X_2 Y_2} \\ & - (AA\,CA + AB\,CB + AC\,CC)\,\overline{X_2 Z_2} \\ & - (BA\,CA + BB\,CB + BC\,CC)\,\overline{Y_2 Z_2} \;) \\ & - (\overline{X_1} - AA\,\overline{X_2} - BA\,\overline{Y_2} - CA\,\overline{Z_2})^2 \\ & - (\overline{Y_1} - AB\,\overline{X_2} - BB\,\overline{Y_2} - CB\,\overline{Z_2})^2 \\ & - (\overline{Z_1} - AC\,\overline{X_2} - BC\,\overline{Y_2} - CC\,\overline{Z_2})^2 \end{aligned}$$

The linear first approximation is now found by partially differentiating this new *Error* with respect to *AA* through *CC*, setting these derivatives to zero and simultaneously solving the system of nine equations (which happen to be linear) so obtained for *AA* through *CC*.

Now we get to the hard part. The matrix obtained in the preceding step is a general 3x3 transform, and not necessarily a true rotation. A true 3D rotation has only three free parameters, not nine. However these three parameters are chosen, some or all of the elements of a rotation matrix that characterizes the same rotation will be non-linear functions of the three parameters.

We have chosen to represent rotations by a three element subset of a quaternion. A quaternion is a four element parameterization of a rotation, but any of the four elements may be derived from the other three (the sum of their squares is unity).

Imagine an arbitrary axis through the origin, described by its angles α, β and γ with the co-ordinate axes. A general rotation can be specified as a twist of an angle θ around such an axis. The quaternion for this rotation is the four element list

$$[P_0, P_x, P_y, P_z] \quad = \quad [\cos\theta/2, \cos\alpha\sin\theta/2, \cos\beta\sin\theta/2, \cos\gamma\sin\theta/2]$$

The half angle components may seem a little peculiar, but the manipulations are simplified because of them. The rotation matrix corresponding to this quaternion is

$$\begin{pmatrix} -1+2P_0^2+2P_x^2 & 2P_xP_y-2P_0P_z & 2P_0P_y+2P_xP_z \\ 2P_xP_y+2P_0P_z & -1+2P_0^2+2P_y^2 & -2P_0P_x+2P_yP_z \\ -2P_0P_y+2P_xP_z & 2P_0P_x+2P_yP_z & -1+2P_0^2+2P_z^2 \end{pmatrix}$$

The P_0 element can be defined positive and equal to $\sqrt{1-P_x^2-P_y^2-P_z^2}$ With this last substitution, the rotation is defined by three independent variables.

Substituting these matrix elements (now involving square roots for P_0) in place of *AA* through *CC* in *Error* creates an expression only a barbarian would inflict upon a reader.

Suffice it to say that the resulting expression is partially differentiated with respect to P_x, P_y and P_z, and the resulting, very non-linear, expressions are equated to zero. Let E represent the barbarous *Error* parameterized by the three quaternion sine elements. Name the three expressions as follows

$$E_x = \frac{\partial E}{\partial P_x}, \quad E_y = \frac{\partial E}{\partial P_y}, \quad E_z = \frac{\partial E}{\partial P_z}$$

The Newton's method iteration is defined by

$$[P_x, P_y, P_z]' \quad = \quad [P_x, P_y, P_z] - [E_x, E_y, E_z] \begin{pmatrix} \frac{\partial E_x}{\partial P_x} & \frac{\partial E_x}{\partial P_y} & \frac{\partial E_x}{\partial P_z} \\ \frac{\partial E_y}{\partial P_x} & \frac{\partial E_y}{\partial P_y} & \frac{\partial E_y}{\partial P_z} \\ \frac{\partial E_z}{\partial P_x} & \frac{\partial E_z}{\partial P_y} & \frac{\partial E_z}{\partial P_z} \end{pmatrix}^{-1}$$

The derivatives in the matrix, as well as E_x through E_z are messy closed form expressions of P_x, P_y and P_z.

The bulk of the code in this part of the program, about a page of solid formulas, was produced automatically by MIT's *Macsyma* symbolic mathematics system [M1].

Path Planning

Exact Shortest Path in Circle Space

The following test program demonstrates an algorithm which finds the guaranteed best tangent path. It was not used in the cart program because of its slow running time and large arrays. An approximate method given in the next section was chosen instead.

The positions and radii of N circular obstacles are given in arrays X, Y and R, indices 1 thru N. X and Y also hold the desired starting and stopping co-ordinates in indices 0 and $N+1$.

The best path is recorded in arrays $PATH\_XI$, $PATH\_YI$, $PATH\_XJ$, $PATH\_YJ$ and $PATH\_R$, indices 1 thru $PATH\_N$. For each index i, $PATH\_XI[i]$ and $PATH\_YI[i]$ are the starting co-ordinates of a tangential segment, $PATH\_XJ[i]$ and $PATH\_YJ[i]$ are the finish. $PATH\_R[i]$ is the radius of the circular arc which connects (and is tangent to) this straight run i, and the one given by $i+1$. If $PATH\_R$ is positive, the connecting circle is to be traversed in a counter-clockwise direction, or clockwise if negative. The center of the circular arc must be determined by analytic geometry from its tangents and

radius.

In the procedure, each obstacle becomes two, one representing a counter-clockwise approach by a tangent (positive index), and the other a clockwise approach (negative index).

```
BEGIN "AVOID1" comment test progrsm for shortest path alg;

REQUIRE "DDHDR.SAI[GRA,HPM]" SOURCE_FILE;
DEFINE PI="3.14159265",TWOPI="(2*PI)";

DEFINE METER="3.2808399", XPOINTS=2, CART_RADIUS=3, TURN_RADIUS=9;
INTEGER I,J,K,N,NREAL,CH; REAL LEN;
REAL ARRAY X,Y,R[0:200];
INTEGER PATH_N;
REAL ARRAY PATH_XI,PATH_YI,PATH_XJ,PATH_YJ,PATH_RI[0:200];

DDINIT; SCREEN(-.01,-.01,10.01,7.01);
CH←GDDCHN(-1); DPYUP(CH); SHOW(CH,'401);

WHILE TRUE DO
   BEGIN
   DDINIT; LITEN;
   FOR I←0 STEP 1 UNTIL 7 DO LINE(0,I,10,I);
   FOR J←0 STEP 1 UNTIL 10 DO LINE(J,0,J,7);

   X[0]←0; Y[0]←0;  R[0]←1.05-8;  N←0;
   FOR J←1 STEP 1 UNTIL 10 DO
     BEGIN N←N+1; R[N]←(RAN(0)+.4)*.5;
     X[N]←(10-2*R[N])*RAN(0)+R[N]; Y[N]←(7-2*R[N])*RAN(0)+R[N]; END;

   FOR J←1 STEP 1 UNTIL 10 DO
      BEGIN REAL P; N←N+1; R[N]←(RAN(0)+.4)*.5;
      X[N]←(10-2*R[N])*(P←RAN(0))+R[N];
      Y[N]←(7-2*R[N])*(P←.4+RAN(0)*.6)+R[N]; END;

   INVEN;
   NREAL←N;

   FOR I←1 STEP 1 UNTIL N DO
      BEGIN
      ELLIPS(X[I]-R[I]+.05,Y[I]-R[I]+.05,
             X[I]+R[I]-.05,Y[I]+R[I]-.05);
      FNTPOS(X[I],Y[I]);
      FNTEXT(IF I≤0 THEN -7 ELSE -13,-8,0,CVS(I));
```

```
    END;
DPYUP(CH);

X[N+1]←10; Y[N+1]←7; R[N+1]←1.0@-8;

    BEGIN
    DEFINE INF="1.0@20";
    STRING PROCEDURE CVH(REAL X);
    RETURN(IF X≥INF THEN "INF" ELSE IF X≤-INF THEN "-INF"
           ELSE CVF(X));

    INTEGER L,PL;
    INTEGER ARRAY PERM,PATH[-N:N+1,-N:N+1];
    REAL ARRAY BEST,XI,YI,XJ,YJ,RI,DIST[-N-1:N+1,-N-1:N+1];
    REAL IR;

    REAL PROCEDURE DISTA(INTEGER I,J);
    IF DIST[I,J]≠-INF THEN RETURN(DIST[I,J]) ELSE
    IF DIST[-J,-I]≠-INF THEN
       BEGIN
       DIST[I,J]←DIST[-J,-I];
       XJ[I,J]←XI[-J,-I];   YJ[I,J]←YI[-J,-I];
       XI[I,J]←XJ[-J,-I];   YI[I,J]←YJ[-J,-I];
       RETURN(DIST[I,J]);
       END
    ELSE IF ABS(I)=ABS(J) THEN RETURN(DIST[I,J]←INF) ELSE
       BEGIN
       REAL IX,IY,JX,JY;
       REAL A,D,D2,DX,DY,RA,RB,XA,XB,YA,YB;
       XA←X[ABS(I)]; YA←Y[ABS(I)];  RA←R[ABS(I)];
       IF I<0 THEN RA←-RA;
       XB←X[ABS(J)]; YB←Y[ABS(J)];  RB←R[ABS(J)];
       IF J<0 THEN RB←-RB;
       DX←XB-XA; DY←YB-YA; D2←DX*DX+DY*DY; D←SQRT(D2);
       DX←DX/D; DY←DY/D;
       IF D2=0 ∨ ABS(A←(RA-RB)/D)≥1 THEN RETURN(DIST[I,J]←INF)
       ELSE
          BEGIN
          REAL B,PAR,PER,DISTIJ;
          B←SQRT(1-A*A); DISTIJ←D*B; PAR←DX*A+DY*B; PER←DY*A-DX*B;
          IX←XA+RA*PAR; IY←YA+RA*PER; JX←XB+RB*PAR; JY←YB+RB*PER;
          XI[I,J]←IX; YI[I,J]←IY; XJ[I,J]←JX; YJ[I,J]←JY;
          DX←JX-IX; DY←JY-IY; D2←DX*DX+DY*DY; D←SQRT(D2);
          DX←DX/D; DY←DY/D;
          FOR K←1 STEP 1 UNTIL (IF I=0∨J=0 THEN N ELSE NREAL) DO
             BEGIN IF K≠ABS(I) ∧ K≠ABS(J) THEN
                BEGIN  REAL XC,YC,RC,WAY,RC2,DYC,DXC;
```



```
                XC←X[K]; YC←Y[K]; RC←R[K]; DXC←XC-IX; DYC←YC-IY;
                IF ABS(DX*DYC-DY*DXC)≤RC THEN
                   BEGIN  WAY←DY*DYC+DX*DXC;  RC2←RC*RC;
                   IF (WAY≥0∧WAY≤D)∨DXC*DXC+DYC*DYC≤RC2
                      ∨(JX-XC)↑2+(JY-YC)↑2≤RC2
                   THEN RETURN(DIST[I,J]←INF); END; END; END;
             RETURN(DIST[I,J]←DISTIJ);
             END;
       END;

REAL PROCEDURE ARCLEN(INTEGER OBP,OBS,OBN);
    BEGIN REAL TH1,TH2,DTH,XC,YC,RC,RET1,RET2,X1,Y1,X2,Y2;
          INTEGER O;
    IF OBS=0 OR OBS=N+1 THEN RETURN(0);
    O←ABS(OBS); RC←R[O];
    X1←XJ[OBP,OBS]; Y1←YJ[OBP,OBS];
    X2←XI[OBS,OBN]; Y2←YI[OBS,OBN];
    IR←RC*(IF OBS<0 THEN -1 ELSE 1); XC←X[O]; YC←Y[O];
    TH2←ATAN2(Y2-YC,X2-XC);   TH1←ATAN2(Y1-YC,X1-XC);
    IF IR<0 THEN TH1'TH2;
    DTH←TH2-TH1;  IF DTH<0 THEN DTH←DTH+TWOPI;
    RET1←RC*DTH;  RET2←RC*(TWOPI-DTH);
    TH2←TH2-TH1;
    WHILE TH2<0 DO TH2←TH2+TWOPI;
    WHILE TH2≥TWOPI DO TH2←TH2-TWOPI;
    FOR K←1 STEP 1 UNTIL NREAL DO
       BEGIN "OBSTST"
       IF K≠O THEN
          BEGIN
          REAL RB,XB,YB,SEP;
          XB←X[K]; YB←Y[K]; RB←R[K];
          SEP←SQRT((XB-XC)↑2+(YB-YC)↑2);
          IF SEP<RB+RC ∧ SEP>RC-RB THEN    comment overlap;
          IF SEP≤RB-RC THEN RETURN(INF) ELSE
             comment total occlusion;
             BEGIN   comment the hard part, a partial overlap;
             REAL X2,Y2,DYX,DYXR,RT,XI1,YI1,XI2,YI2,TI1,TI2;
             X2←XB-XC; Y2←YB-YC, DYX←X2↑2+Y2↑2;
             DYXR←DYX-RB↑2+RC↑2;
             RT←4*DYX*RC↑2-DYXR↑2;
             IF RT<0 THEN
                BEGIN PRINT("RT neg in ARCLEN ",RT,"18A'12);
                      RT←0; END;
             RT←SQRT(RT);
             comment intersect two obstacles;
             XI1←(DYXR*X2+Y2*RT)/(2*DYX)+XC;
             YI1←(DYXR*Y2-X2*RT)/(2*DYX)+YC;
```

```
            XI2←(DYXR*X2-Y2*RT)/(2*DYX)+XC;
            YI2←(DYXR*Y2+X2*RT)/(2*DYX)+YC;
            TI1←ATAN2(YI1-YC,XI1-XC); TI2←ATAN2(YI2-YC,XI2-XC);
            TI1←TI1-TN1; TI2←TI2-TN1;
            WHILE TI1<0 DO TI1←TI1+TWOPI;
            WHILE TI1≥TWOPI DO TI1←TI1-TWOPI;
            WHILE TI2<0 DO TI2←TI2+TWOPI;
            WHILE TI2≥TWOPI DO TI2←TI2-TWOPI;
            IF TI2<TI1 THEN RET1←RET2←INF;
                        comment occludes both directions;
            IF TI1≤TH2 THEN RET1←INF;
                        comment occludes forward direction;
            IF TI2≥TH2 THEN RET2←INF;
                        comment occludes reverse direction;
            IF RET1=INF∨RET2=INF THEN
               BEGIN LITEN; LINE(XI1,YI1,XI2,YI2); END;
            IF RET1=INF∧RET2=INF THEN DONE "OBSTST";
            END;
         END;
      END "OBSTST";
   IF RET2<RET1 THEN BEGIN RET2←RET1; IR←-IR; END;
   RETURN(RET1);
   END;

ARRCLR(DIST,-INF);

FOR I←-N STEP 1 UNTIL N+1 DO FOR J←-N STEP 1 UNTIL N+1 DO
    BEGIN BEST[J,I]←INF; PERM[J,I]←FALSE; PATH[J,I]←0; END;
FOR J←-N STEP 1 UNTIL N+1 DO
   BEGIN BEST[J,0]←0; PERM[J,J]←PERM[J,0]←TRUE;
         XJ[J,0]←X[0]; YJ[J,0]←Y[0]; END;

PL←0, L←0;
WHILE L≠N+1 DO
   BEGIN "ROUTE" REAL BMV,BL; BL←BEST[PL,L];
   FOR I←-N STEP 1 UNTIL N+1 DO IF ¬PERM[L,I] THEN
      BEGIN REAL DLI,BI,
      DLI←DISTA(L,I), BI←BEST[L,I], DLI←BL+DLI,
      IF DLI<BI ∧ (DLI←DLI+ARCLEN(PL,L,I))<BI THEN
         comment sets IR,
         BEGIN RI[L,I]←IR, BEST[L,I]←DLI, PATH[L,I]←PL, END,

      END,
   BMV←INF,
   FOR I←-N STEP 1 UNTIL N+1 DO FOR J←-N STEP 1 UNTIL N+1 DO
      IF ¬PERM[I,J]∧BEST[I,J]<BMV THEN
         BEGIN BMV←BEST[I,J], PL←I, L←J, END,
```



```
      IF BMV≥INF THEN
         BEGIN PRINT("No path!!!","15A'12); DONE "ROUTE"; END;
      PERM[PL,L]←TRUE;
      END "ROUTE";

   PATH_N←0;
   PRINT("[0] ",BEST[0,0],"  [",N+1,"] ",BEST[PL,L],"15A'12);
   DO   comment  record the best path;
      BEGIN  INTEGER T; PATH_N←PATH_N+1;
      PATH_XI[PATH_N]←XI[PL,L]; PATH_YI[PATH_N]←YI[PL,L];
      PATH_XJ[PATH_N]←XJ[PL,L]; PATH_YJ[PATH_N]←YJ[PL,L];
      PATH_RI[PATH_N]←RI[PL,L]; T←PATH[PL,L]; L←PL; PL←T; END
   UNTIL L=0;
   END;

LITEN; LEN←0;
FOR I←PATH_N STEP -1 UNTIL 1 DO
   BEGIN
   LINE(PATH_XI[I],PATH_YI[I],PATH_XJ[I],PATH_YJ[I],PATH_N+1-I);
   LEN←LEN+
     SQRT((PATH_XI[I]-PATH_XJ[I])↑2+(PATH_YI[I]-PATH_YJ[I])↑2);

   ELLIPS(PATH_XJ[I]-.02,PATH_YJ[I]-.02,
          PATH_XJ[I]+.02,PATH_YJ[I]+.02);
   IF I>1 THEN
      BEGIN
      REAL X1,Y1,X2,Y2,DX1,DY1,DX2,DY2,XC,YC,R,XL,YL,DTH,SD,CD;
      INTEGER NS,IS;
      X1←PATH_XJ[I];   Y1←PATH_YJ[I];
      X2←PATH_XI[I-1]; Y2←PATH_YI[I-1];
      DX1←PATH_XJ[I]-PATH_XI[I];  DY1←PATH_YJ[I]-PATH_YI[I];
      DX2←PATH_XJ[I-1]-PATH_XI[I-1];
      DY2←PATH_YJ[I-1]-PATH_YI[I-1];
      XC←(DY1*(DY2*Y2+DX2*X2)-DY2*(DY1*Y1+DX1*X1))
                                          /(DX2*DY1-DX1*DY2);
      YC←(DX2*(DY1*Y1+DX1*X1)-DX1*(DY2*Y2+DX2*X2))
                                          /(DX2*DY1-DX1*DY2);
      DTH←ATAN2(Y2-YC,X2-XC)-ATAN2(Y1-YC,X1-XC);  R←PATH_RI[I-1];
      IF R<0 THEN DTH←-DTH, IF DTH<0 THEN DTH←DTH+TWOPI;

      LEN←LEN+ABS(R*DTH);
      NS←5*R*DTH, NS←NS←(IF NS<0 THEN -1 ELSE 1);
      SD←SIN(DTH/NS); CD←COS(DTH/NS);
      XL←X1; YL←Y1;
      FOR IS←1 STEP 1 UNTIL ABS(NS) DO
         BEGIN REAL X3,Y3,
         X3←XC+CD*(XL-XC)-SD*(YL-YC),
```

```
            Y3←YC+CD*(YL-YC)+BD*(XL-XC),
            LINE(XL,YL,X3,Y3); XL←X3; YL←Y3;
            END,
         END;
      END,
   DPYUP(CH),
   PRINT("LENGTH ",LEN),
   INCHWL;
   END,
END "AVOID1";
```

Approximate Shortest Path in Circle Space

A version of the following code was used by the cart program. It differs from the exact algorithm in that it does not search as many possibilities, and is sometimes fooled by the difference in distances caused by arrival at an obstacle from different directions. In typical 100 obstacle courses, it gives answers trivially different from the exact ones about 5% of the time.

```
BEGIN "AVOID"
   comment test program for approximate shortest path alg;
REQUIRE "DDHDR SAI[GRA,HPW]" SOURCE_FILE;
DEFINE PI="3.14159265",TWOPI="(2*PI)";

DEFINE METER="3.2808399", XPOINTS=2, CART_RADIUS=3, TURN_RADIUS=9;
INTEGER I,J,K,N,NREAL,CH, REAL LEN;
REAL ARRAY X,Y,R[0:200];
INTEGER PATH_N;
REAL ARRAY PATH_XI,PATH_YI,PATH_XJ,PATH_YJ,PATH_RI[0:200];

PROCEDURE CONSOLIDATE;
     comment this procedure can be called repeatedly
             to coalesce overlapping obstacles. It was not
             used in the final program;
   BEGIN
   INTEGER I,J;

   FOR I←1 STEP 1 UNTIL N-1 DO
      BEGIN
      REAL RC,XC,YC,RB,XB,YB,SEP;
```



```
      FOR J←I+1 STEP 1 UNTIL N DO
         BEGIN
         RC←R[I]; XC←X[I]; YC←Y[I];
         RB←R[J]; XB←X[J]; YB←Y[J];
         SEP←SQRT((XB-XC)↑2+(YB-YC)↑2);
         IF RC<RB THEN BEGIN RC←RB; XB←XC; YB←YC; END;
         IF SEP>(RB+RC)*0.8 THEN comment no contact; ELSE
         IF SEP<RC-RB THEN
            BEGIN
            comment total overlap;
            X[I]←XC; Y[I]←YC; R[I]←RC;
            X[J]←X[N]; Y[J]←Y[N]; R[J]←R[N]; N←N-1; J←J-1;
            END
         ELSE
            BEGIN comment the hard part, a partial overlap;
            REAL XD,YD,RD,K,POKE;
            POKE←RB+SEP-RC;
            K←1/2+(RC-RB)/(2*SEP);
            XD←XC*K+XB*(1-K);
            YD←YC*K+YB*(1-K);
            RD←(RC + RB + SEP)/2 - POKE/8;
            X[I]←XD; Y[I]←YD; R[I]←RD;
            X[J]←X[N]; Y[J]←Y[N]; R[J]←R[N]; N←N-1; J←J-1;
            END;
         END;
      END;
   END;

DDINIT; SCREEN(-.01,-.01,10.01,7.01); CH←GDDCHN(-1);
DPYUP(CH); SHOW(CH,"401");

WHILE TRUE DO
   BEGIN
   DDINIT; LITEN;
   FOR I←0 STEP 1 UNTIL 7 DO LINE(0,I,10,I);
   FOR J←0 STEP 1 UNTIL 10 DO LINE(J,0,J,7);

   X[0]←0; Y[0]←0;  R[0]←1.00-8;  N←0;
   FOR J←1 STEP 1 UNTIL 10 DO
      BEGIN N←N+1; R[N]←(RAN(0)+.4)* 5;
      X[N]←(10-2*R[N])*RAN(0)+R[N];
      Y[N]←(7-2*R[N])*RAN(0)+R[N]; END;

   FOR J←1 STEP 1 UNTIL 10 DO
      BEGIN REAL P; N←N+1; R[N]←(RAN(0)+.4)* 5;
      X[N]←(10-2*R[N])*(P←RAN(0));R[N];
      Y[N]←(7-2*R[N])*(P←.4+RAN(0)* 6)+R[N]; END;
```

```
INVEN;

NREAL←N;

 FOR I←1 STEP 1 UNTIL N DO
    ELLIPS(X[I]-R[I]+.05,Y[I]-R[I]+.05,
           X[I]+R[I]-.05,Y[I]+R[I]-.05);
 DPYUP(CH);

 X[N+1]←10; Y[N+1]←7; R[N+1]←1.00-8;

    BEGIN
    DEFINE INF="1 0020";
    INTEGER LATEST;
    INTEGER ARRAY PERM,PATH[-N:N+1];
    REAL ARRAY BEST,XI,YI,XJ,YJ,RI[-N:N+1];
    REAL IX,IY,JX,JY,IR;

    REAL PROCEDURE DISTA(INTEGER I,J);
    IF ABS(I)=ABS(J) THEN RETURN(INF) ELSE
       BEGIN
       REAL A,D,D2,DX,DY,RA,RB,XA,XB,YA,YB;
       XA←X[ABS(I)]; YA←Y[ABS(I)];
       RA←R[ABS(I)]; IF I<0 THEN RA←-RA;
       XB←X[ABS(J)]; YB←Y[ABS(J)];
       RB←R[ABS(J)]; IF J<0 THEN RB←-RB;
       DX←XB-XA; DY←YB-YA; D2←DX*DX+DY*DY;
       D←SQRT(D2); DX←DX/D; DY←DY/D;
       IF D2=0 ∨ ABS(A←(RA-RB)/D)≥1 THEN RETURN(INF) ELSE
          BEGIN
          REAL B,PAR,PER,DISTIJ;
          B←SQRT(1-A*A); DISTIJ←D*B; PAR←DX*A+DY*B; PER←DY*A-DX*B;
          IX←XA+RA*PAR; IY←YA+RA*PER; JX←XB+RB*PAR; JY←YB+RB*PER;
          DX←JX-IX; DY←JY-IY; D2←DX*DX+DY*DY;
          D←SQRT(D2); DX←DX/D; DY←DY/D;
          FOR K←1 STEP 1 UNTIL (IF I=0∨J=0 THEN N ELSE NREAL) DO
             BEGIN IF K≠ABS(I) ∧ K≠ABS(J) THEN
                BEGIN REAL XC,YC,RC,WAY,RC2,DYC,DXC;
                XC←X[K]; YC←Y[K]; RC←R[K]; DXC←XC-IX; DYC←YC-IY;
                IF ABS(DX*DYC-DY*DXC)≤RC THEN
                   BEGIN WAY←DY*DYC+DX*DXC; RC2←RC*RC;
                   IF (WAY≥0∧WAY≤D)∨DXC*DXC+DYC*DYC≤RC2
                      ∨(JX-XC)↑2+(JY-YC)↑2≤RC2
                   THEN RETURN(INF); END; END; END;
          RETURN(DISTIJ);
          END;
```



```
    END;

REAL PROCEDURE ARCLEN(REAL X1,Y1,X2,Y2; INTEGER OBS);
   BEGIN REAL TH1,TH2,DTH,XC,YC,RC,RET1,RET2; INTEGER O;
   O←ABS(OBS); RC←R[O];
   IR←RC*(IF OBS<0 THEN -1 ELSE 1); XC←X[O]; YC←Y[O];
   TH2←ATAN2(Y2-YC,X2-XC);   TH1←ATAN2(Y1-YC,X1-XC);
   IF IR<0 THEN TH1↔TH2;
   DTH←TH2-TH1;  IF DTH<0 THEN DTH←DTH+TWOPI;
   RET1←RC*DTH;  RET2←RC*(TWOPI-DTH);
   TH2←TH2-TH1;
   WHILE TH2<0 DO TH2←TH2+TWOPI;
   WHILE TH2≥TWOPI DO TH2←TH2-TWOPI;
   FOR K←1 STEP 1 UNTIL NREAL DO IF K≠O THEN
      BEGIN
      REAL RB,XB,YB,SEP;
      XB←X[K]; YB←Y[K]; RB←R[K];
      SEP←SQRT((XB-XC)↑2+(YB-YC)↑2);
      IF SEP<RB+RC ∧ SEP>RC-RB THEN     comment overlap;
      IF SEP≤RB-RC THEN RETURN(INF) ELSE
      comment total occlusion;
         BEGIN     comment the hard part, a partial overlap;
         REAL X2,Y2,DYX,DYXR,RT,XI1,YI1,XI2,YI2,TI1,TI2;
         X2←XB-XC; Y2←YB-YC;
         DYX←X2↑2+Y2↑2; DYXR←DYX-RB↑2+RC↑2;
         RT←4*DYX*RC↑2-DYXR↑2;
         IF RT<0 THEN BEGIN
            PRINT("RT neg in ARCLEN ",RT,"18&"12); RT←0; END;
         RT←SQRT(RT);
         comment intersect two obstacles;
         XI1←(DYXR*X2+Y2*RT)/(2*DYX)+XC;
         YI1←(DYXR*Y2-X2*RT)/(2*DYX)+YC;
         XI2←(DYXR*X2-Y2*RT)/(2*DYX)+XC;
         YI2←(DYXR*Y2+X2*RT)/(2*DYX)+YC;
         TI1←ATAN2(YI1-YC,XI1-XC);
         TI2←ATAN2(YI2-YC,XI2-XC);
         TI1←TI1-TH1; TI2←TI2-TH1;
         WHILE TI1<0 DO TI1←TI1+TWOPI;
         WHILE TI1≥TWOPI DO TI1←TI1-TWOPI;
         WHILE TI2<0 DO TI2←TI2+TWOPI;
         WHILE TI2≥TWOPI DO TI2←TI2-TWOPI;
         IF TI2<TI1 THEN RET1←RET2←INF;
            comment occludes both directions;
         IF TI1≤TH2 THEN RET1←INF;
            comment occludes forward direction;
         IF TI2≥TH2 THEN RET2←INF;
            comment occludes reverse direction;
```

```
            IF RET1=INF∨RET2=INF THEN
               BEGIN LITEN; LINE(XI1,YI1,XI2,YI2); END;
            END;
         END;
      IF RET2<RET1 THEN
         BEGIN RET2←RET1; IR←-IR; END;  RETURN(RET1);
      END;

   FOR I←-N STEP 1 UNTIL N+1 DO
      BEGIN BEST[I]←INF; PERM[I]←FALSE; PATH[I]←0; END;
   BEST[0]←0; PERM[0]←TRUE; LATEST←0;  XJ[0]←X[0]; YJ[0]←Y[0];
   WHILE LATEST≠N+1 DO
      BEGIN "ROUTE" INTEGER SWI; REAL SWV,XJL,YJL,BL;
      SWI←0; SWV←INF;
      BL←BEST[LATEST]; XJL←XJ[LATEST]; YJL←YJ[LATEST];
      FOR I←-N STEP 1 UNTIL N+1 DO IF ¬PERM[I] THEN
         BEGIN  REAL DLI,BI;
         DLI←DISTA(LATEST,I); BI←BEST[I]; DLI←BL←DLI;
            comment sets IX,IY,JX,JY;
         IF DLI<BI ∧ (DLI←DLI+ARCLEN(XJL,YJL,IX,IY,LATEST))<BI
         THEN              comment sets IR;
            BEGIN XI[I]←IX; YI[I]←IY; XJ[I]←JX; YJ[I]←JY;
            RI[I]←IR; BEST[I]←BI←DLI; PATH[I]←LATEST; END;
         IF BI<SWV THEN BEGIN SWV←BI; SWI←I; END;
         END;
      IF SWV≥INF THEN
         BEGIN PRINT("No path!!!",'15&'12); DONE "ROUTE"; END;
      LATEST←SWI; PERM[LATEST]←TRUE;
      END "ROUTE";

   PATH_N←0;  K←N+1;
   PRINT("[0] ",BEST[0]," [N+1] ",BEST[N+1],'15&'12);
   WHILE K≠0 DO   comment  record the best path;
      BEGIN INTEGER L; L←PATH[K];  PATH_N←PATH_N+1;
      PATH_XI[PATH_N]←XI[K]; PATH_YI[PATH_N]←YI[K];
      PATH_XJ[PATH_N]←XJ[K]; PATH_YJ[PATH_N]←YJ[K];
      PATH_RI[PATH_N]←RI[K];  K←L; END;
   END;

LITEN; LEN←0;
FOR I←PATH_N STEP -1 UNTIL 1 DO
   BEGIN
   LINE(PATH_XI[I],PATH_YI[I],PATH_XJ[I],PATH_YJ[I],PATH_N+1-I);
   LEN←LEN+
   SQRT((PATH_XI[I]-PATH_XJ[I])↑2+(PATH_YI[I]-PATH_YJ[I])↑2);
   ELLIPS(PATH_XJ[I]-.02,PATH_YJ[I]-.02,
          PATH_XJ[I]+.02,PATH_YJ[I]+.02);
```



```
      IF I>1 THEN
         BEGIN
         REAL X1,Y1,X2,Y2,DX1,DY1,DX2,DY2,XC,YC,R,XL,YL,DTH,SD,CD;
         INTEGER NS,IS;
         X1←PATH_XJ[I];   Y1←PATH_YJ[I];
         X2←PATH_XI[I-1], Y2←PATH_YI[I-1];
         DX1←PATH_XJ[I]-PATH_XI[I];  DY1←PATH_YJ[I]-PATH_YI[I];
         DX2←PATH_XJ[I-1]-PATH_XI[I-1];
         DY2←PATH_YJ[I-1]-PATH_YI[I-1];
         XC←(DY1×(DY2×Y2+DX2×X2)-DY2×(DY1×Y1+DX1×X1))
                                                 /(DX2×DY1-DX1×DY2);
         YC←(DX2×(DY1×Y1+DX1×X1)-DX1×(DY2×Y2+DX2×X2))
                                                 /(DX2×DY1-DX1×DY2);
         DTH←ATAN2(Y2-YC,X2-XC)-ATAN2(Y1-YC,X1-XC); R←PATH_RI[I-1];
         IF R<0 THEN DTH←-DTH; IF DTH<0 THEN DTH←DTH+TWOPI;

         LEN←LEN+ABS(R×DTH);
         NS←5×R×DTH; NS←NS←(IF NS<0 THEN -1 ELSE 1);
         SD←SIN(DTH/NS); CD←COS(DTH/NS);
         XL←X1; YL←Y1;
         FOR IS←1 STEP 1 UNTIL ABS(NS) DO
            BEGIN REAL X3,Y3;
            X3←XC←CD×(XL-XC)-SD×(YL-YC);
            Y3←YC←CD×(YL-YC)←SD×(XL-XC);
            LINE(XL,YL,X3,Y3); XL←X3; YL←Y3;
            END;
         END;
      END;
   DPYUP(CH);
   PRINT("LENGTH ",LEN);
   INCHWL;
   END;
END "AVOID";
```

Spinoffs

Guide to Data Disc Graphics Routines

to use: REQUIRE "DDHDR.SAI[GRA,HPM]" SOURCE_FILE;

uses: DDFAI.REL[GRA], DDSAI.REL[GRA]

DDINIT initialise the DD buffer

SCREEN(REAL XLO, YLO, XHI, YHI) declare the full screen dimensions

SCREEM(REAL XLO, YLO, XHI, YHI) returns the full screen dimensions

DRKEN cause subsequent outputs to be dark

LITEN cause them to be light

INVEN cause them to invert the state of things

DOT(X, Y, THICK(0)) display a point at X, Y

LINE(X1, Y1, X2, Y2, THICK(0)) display a line

RECTAN(X1, Y1, X2, Y2) fill in rectangle

ELLIPS(X1, Y1, X2, Y2) fill ellipse bounded by X1, Y1, X2, Y2

POLYGO(N, X[1:N], Y[1:N]) fill in a polygon (concave and star ok)

TXTPOS(X, Y, XS, YS, DXS(0), DYS(0)) Position text start at X, Y with general linear transf. First char's corners will be (X, Y), (X+DXS, Y+YS), (X+XS+DXS, Y+YS+DYS), (X+XS, Y+DYS). To make normal horisontal text of characters W wide and H high, do

$$TXTPOS(X,Y,W,H,0,0)$$

or just $TXTPOS(X,Y,W,H)$. To make sideways text lying on its left side, do

$$TXTPOS(X,Y,0,0,-H,W)$$

To make text on its right side (reading downwards), do

$$TXTPOS(X,Y,0,0,H,-W)$$

To make text *rotated from* the horisontal by an arbitrary angle T, do

$$TXTPOS(X,Y,W\cos T,H\cos T,-H\sin T,W\sin T)$$

To make horisontal, italicised, text, do

$$TXTPOS(X,Y,W,H,W/2,0)$$

I leave the other possibilities to you. Consider XS and YS the main diagonal of a 2 x 2 matrix that takes a straight, horisontal, prototype character into the skewed, rotated one actually drawn. DXS and DYS are the off-diagonal elements of that matrix. Text can thus be rotated, *italicized and reflected*.

TEXT(STRING) vector text, positioned by previous TXTPOS

TEXTD(STRING) dot char text. Look better for tiny, inven

CHAN = GDDCHN(CHAN) get a DD channel (-1 for any, failure)

RDDCHN(CHAN) release a channel

ERASE(CHAN) clear a DD channel (-1 means yours

DPYUP(CHAN, BUFFER(-1)) Deposit the buffer on DD channel CHAN

PJUP send buffer to MOS display on PDP11

SHOW(CHAN, LIN(-1)) video switch LIN (-1 for yours) to CHAN

SHOWA(CHAN, LIN(-1)) add CHAN to LIN

SHOWS(CHAN, LIN(-1)) subtract CHAN from LIN

PPPOS(YLO, YHI) position page printer between YLO, YHI

XGPUP(SIZE) send the DD buffer to the XGP, SIZE= −5 to +5

XGPQUE(SIZE) like XGPUP, but creates a new job to do XGPing

DDFONT(X1, Y1, X2, Y2, FONTFIL, CHAR("A"), BASE(0), LKERN(0), RKERN(0)) insert DD buffer bounded by X1, Y1, X2, Y2 into font file. DDPAK(I, LBUF, J1, J2); pack scanline I bet. J1, J2 into LBUF 36 bit/word

CHAN = SYNMAP(ORDER) which channels are the video synthesizer's. ORDER=0 is most significant + or - 1 next, etc.

SUCCESS = MAPSET(F) set video inten table. F(X) is real [0, 1]⇒[0, 1]

SUCCESS = MAPMON(P) set v.i.t. to a monotonic function X^P

SUCCESS = MAPGRY(P) set v.i.t. to a gray coded monotonic function X^P

LINSCN(N, MAP, DT, LINENO) scan channels in MAP at rate DT

MAPSCN(N, MAP, DT, LINENO) *LINSCN, but VDS bit maps in MAP*

SCNOFF turn off the scanning (in SW mode)

SCNFRZ SCNOFF, but doesn't restore screen

SCNINC(INC) change the stepsize in the scan

DDSTOR(DDARRAY) store the buffer in an array

DDLOAD(DDARRAY) load the buffer from an array

DDOR(DDARRAY) or an array into the buffer

DDAND(DDARRAY) and the buffer with an array

DDEXCH(DDARRAY) exchange an array with the buffer

GETDDF(FILENAME) load the buffer from a file

PUTDDF(FILENAME) save the buffer in a file

GETMIT(FILENAME) load the buffer from a file

PUTMIT(FILENAME) save the buffer in a file

FNTSEL(FONT#, "FONTNAME") define a font #

FNTPOS(XP, YP, XS(1), YS(1), DXS(0), DYS(0)) position & transform fonted text

FNTEXT(X, Y, F, TXT) deposit fonted text

FNTLIN(X1, Y1, X2, Y2, THICK(0)) draw line in FNTPOS co-ords

FNTDOT(X1, Y1, THICK(0)) dot in FNTPOS co-ords

FNTREC(X1, Y1, X2, Y2) rectangle

FNTELL(X1, Y1, X2, Y2) ellipse

FNTPOL(N, X1[1:N], Y1[1:N]) polygon

DDSIZ a constant, the number of words in a DD array

Arguments followed by a value in parens default to the bracketed value.

The absolute physical screen dimension are 512 pixels in X by 481 in Y. TEXTD characters are 6 logical pixels in X by 10 in Y, spaces included.

Regarding TXTPOS parameters, note that

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} W\cos(ccwrot) & H(\cos(ccwrot)\tan(cwslant) - \sin(ccwrot)) \\ W\sin(ccwrot) & H(\sin(ccwrot)\tan(cwslant) + \cos(ccwrot)) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} XS & DXS \\ DYS & YS \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

thus

$$XS = W\cos(ccwrot)$$
$$DXS = H(\cos(ccwrot)\tan(cwslant) - \sin(ccwrot))$$
$$DYS = W\sin(ccwrot)$$
$$YS = H(\sin(ccwrot)\tan(cwslant) + \cos(ccwrot))$$

It might be useful to introduce a parameterization that accepts "corner, width, height, rot. and slant angles (degrees)":

$$TP(X, Y, W, H, CCWROT, CWSLANT)$$

Guide to GOD Graphics Routines

to use: REQUIRE "GRAHDR.SAI[GOD,HPM]" SOURCE_FILE;

uses: GRASAI.REL[GOD]

The graphics routines on [GOD,HPM] are superficially similar to the ones on [GRA,HPM], but provide an extended device-independent graphics service. To make a drawing one or more devices (graphics servers) must be activated to receive graphics commands. Currently there are servers for Data discs, the XGP and the video synthesizer. In addition the graphics commands can be written into files (called GOD files), which can be displayed later with system programs called DDJOB, XGPJOB and SYNJOB. GOD files can be called as subroutines in graphics programs. GOD files can also be fed to a program called TSTJOB which makes a SAIL program that, if run, recreates the GOD file given to the TSTJOB. Editing the SAIL program provides a way of modifying existing GOD files.

Diagrams in the form of GOD files can be included in documents by the graphics escape features of XGPSYG.

The following calls are provided in addition to the ones in the GRA package.

JOBID=DDJOB create a server that draws on Data Disc displays.

JOBID=XGPJOB create a graphics server that will output on the XGP.

JOBID=SYNJOB(HIG(480),WID(512),BIT(9)) make a server that produces gray scale renditions of a drawing.

JOBID=FILJOB("FILENAME") make a graphics server that writes graphics commands into a GOD file.

JOBID=TSTJOB create a server that prints graphics commands as a SAIL

program.

JOBID=QUASH(JOBID) temporarily deafen a server.It still exists but will not act on subsequent display commands.

JOBID=INCITE(JOBID) reactivate a server. Undoes the effect of a QUASH.

JOBID = KILJOB(INTEGER JOBID) kill a server. Closes the file opened by a FILJOB, simply expunges all other server types.

JOBID = DETJOB(JOBID,GRAFILE) detach a server, but give it a graphics file to process first. Server is lost to main program.

GRAFIL(GRAFILE) tell currently active servers to begin processing a graphics file. Afterwards they will be ready to listen to main program again.

FNTSEL(FONT#,FONTNAME) select a font, and give it a number (0-127).

FNTPOS(XP,YP,XS(1),YS(1),DXS(0),DYS(0)) position a font pixel grid at XP, YP in SCREEN co-ordinates, with transformation set by other four *parameters. 1, 1, 0, 0 results* in undistorted characters.

FNTEXT(X,Y,FONT#,TEXT) deposit TEXT in the indicated font with lower left corner offset (X,Y) font pixels (as distorted by FNTPOS parameters) from the last FNTPOS position.

FNTLIN(X1,Y1,X2,Y2 INTEGER THICK(0)); draw a line between pixel co-ordinates indicated, in grid defined by last FNTPOS.

FNTDOT(X1,Y1,THICK(0)) a dot in FNTPOS co-ordinates.

FNTREC(X1,Y1,X2,Y2) a FNTPOS rectangle.

FNTELL(X1,Y1,X2,Y2) a FNTPOS ellipse.

FNTPOL(N,X[1:N],Y[1:N]) a FNTPOS polygon.

PICFIL(X1,Y1,X2,Y2,PICFILE) insert a picture in the rectangle bounded in X by X1 and X2 and by Y1 and Y2 in Y. On gray scale servers this picture will come out shaded. On binary devices a halftone is produced. PICFILE should be in hand/eye format.

PICFIT(X1,Y1,X2,Y2 STRING FILE); insert a transposed picture in the indicated rectangle.

BUFSIZ= DDSIZ(INTEGER JOBID(-1)) return the display buffer size of the indicated server.

The TYPHDR typesetting extension

to use REQUIRE "TYPHDR.SAI[GOD,HPM]" SOURCE_FILE;

The TYPHDR routines extend the GRAHDR package to permit reasonably convenient two dimensional typesetting of text in mixed fonts, for inclusion in GOD diagrams. The central concept is of a box of a certain size containing a two dimensional assembly of text and graphics. Such a box has an origin, which may be plopped down at a given place in diagram. Some of the commands create new boxes out of text, some by gluing together other ones, and some change the size or the origin of a box.

The following procedures are provided:

FNTSELECT(FONT#,FONTNAME) use this instead of FNTSEL in the GRAHDR.

BOX=JTXT(FONT#,TEXT) return a box containing TEXT in the indicated font

BOX=JCAT(A,B) thru JCAT6(A,B,C,D,E,F) horizontally combine a number of boxes. Origin ends up on left edge of resulting box. Useful for stringing together different fonts.

BOX=JTAC(A,B) thru JTAC7(A,B,C,D,E,F,G) horizontally combine boxes, but leave origin on the right end.

BOX=JBEL(A,B) thru JBEL7(A,B,C,D,E,F,G) vertically stack text boxes. Useful for assembling multiple lines of text.

BOX=JSUB(A,B) make new box that is box A subscripted by box B.

BOX=JEXP(A,B) make box with A superscripted by B.

BOX=JXBP(A,B,C) make box with A sub B super C.

BOX=PADD(DX1,DY1,DX2,DY2,A) padd box A on all four sides (margins).

BOX=SHIF(DX1,DY1,A) shift origin of box A

BOX=JUL(A) make new box with contents of A underlined.

BOX=JSQR(A) make a square root sign around A.

BOX=JDIV(A,B) center A above B and put a division bar between.

BOX=XCENTER(A) center the origin of box A in X.

BOX=YCENTER(A) center A in Y.

BOX=CENTER(A) move origin of A to its center in X and Y.

BOX=RIGHTIFY(A) move origin to right of A.

BOX=LEFTIFY(A) move origin to left of A.

BOX=TOPIFY(A) move origin to top of A.

BOX=BOTTOMIFY(A) move origin to bottom of A.

DEPOSIT(X,Y,A) deposit box A into the diagram such that its origin is X (FNTPOS distorted) pixels to the right and Y pixels above the text position specified by the last FNTPOS.

Guide to vision routines on [VIS,HPM]

PIXHDR.SAI utility routines for getting, saving, moving, etc. pictures

requires: PIXFAI.REL, PIXSAI.REL

PCLN, PCWD, PCBY, PCBYA, LNWD, LNBY, LNBYA, WDBY, WDBI, BYBI, BPTAB, LINTAB where to find things in picture arrays

VALUE = PIXEL(PICTURE, ROW, COLUMN) value of a particular pixel

VALUE = INTREL(PIC, ROW, COL) interpolating PIXEL. ROW, COL, VALUE real.

PUTEL(PICTURE, ROW, COLUMN, VALUE) change a pixel

ADDEL(PICTURE, ROW, COLUMN, VALUE) increment a pixel

ADDIEL(PIC, ROW, COL, VAL) interpolating ADDEL. R, C, V real.

SIZE = PFLDIM(FILENAME) size of array needed for pic file

SIZE = GETPFD(FILENAME, DIM[0:10]) read in parameters of picture

SIZE = GETPFL(FILENAME, PICTURE) read in a pic file

PUTPFL(PICTURE, FILENAME, MODE(1)) write PICTURE into a file. if MODE=2, file is data compressed, otherwise normal

CHAN = OPNPFL(FILENAME, DIM[0:10]) read in parameters of picture, opened for input at 1st scanline

CHAN = CREPFL(DIM[0:10], FILENAME, MODE(1)) write header for PIC file, opened for output at 1st scanline. compressed if *MODE*=2

PFLIN(CHAN, AR[1], NWDS) read next NWDS words from pic into AR

PFLOUT(CHAN, AR[1], NWDS) write next NWDS words to pic from AR

PFLCLS(CHAN) close picture open on channel

SIZE = PIXDIM(HEIGHT, WIDTH, BITS) size of array for HxWxB picture

SIZE = MAKPIX(HEIGHT, WIDTH, BITS, PICTURE) make skeleton HxWxB picture

SIZE = MAKDIM(H, W, B, P[0:10]) make 11 word skeleton, for out of core pix

WIPE(PICTURE, VALUE(0)) make every data word (not byte!)=VALUE

PIXTRN(SRC, TR, DEST) transforms src into dst by array tr. tr is a 3 by 3 real array. For all pixels (y, z) in dest set $(ty, tz, foo) = (y, z, 1)*transform$. if (ty, tz) is in src then $dest(y, z) = src(ty, tz)$

COPPIC(PICTURE1, PICTURE2) copy pic1 into pic2

TILE(PIC1, YL1, XL1, TY, TX, PIC2, YL2, XL2) take piece of size TYxTX at YL1, XL1 in PIC1, deposit at YL2, XL2 in PIC2

SQTILE(PIC1, YL1, XL1, TY, TX, YSQ, XSQ, PIC2, YL2, XL2) a TYxYSQ by TXxXSQ tile from PIC1 with upleft at YL1, XL1 is squished into a TY by TX tile in PIC2, YL2, XL2 upleft The YSQ by XSQ areas in PIC1 are summed and scaled as needed

SATILE(PIC1, YL1, XL1, TY, TX, YSQ, XSQ, PIC2, YL2, XL2) like SQTILE, but 0 samples in PIC1 leave PIC2 unchanged

HAFPIC(PICTURE1, PICTURE2, MAXBIT) reduce pic to half resolution

SHRINK(PICT1, PICT2) squeeze or expand PICT1 into PICT2 pixels are sampled, not interpolated or averaged.

PICADD(PICTURE, PICSUM) add a picture to a picture

PICSUB(PICA, PICDIFF) subtract. PICA-PICDIFF , PICDIFF

PICMUL(PICTURE, PICPRD) multiply pictures. no bounds check.

PICSH(PIC1, PIC2, DIV) every pixel in PIC1/DIV.PIC2

GRAY(PIC) Convert to gray code.

UNGRAY(PIC) Convert back.

RETRY = CAMPIX(CAMERA, YEDGE, XEDGE, PICTURE, SUMS(1), BCLIP(7), TCLIP(0), MAXTRY(20)) read from a camera

NRETRY = CLPADJ(CAM, BLCIP, TCLIP) find optimum clip levels for CAM

NRETRY = TVSNAP(CAM, YEDGE, XEDGE, PIC, BCLIP, TCLIP, NTRY)

NRETRY = TVRAW(CAM, YEDGE, XEDGE, PIC, BCLIP, TCLIP, NTRY) primitive camera routine, used by CAMPIX

TVBTMX(PIC4, PICN, XFRM, INHIBEQ) primitive camera routine, used by CAMPIX

TVBTMY(PIC4, PICN, XFRM, INHIBLE) primitive camera routine, used by CAMPIX

TVBTMZ(PIC4, PICN, XFRM, INHIBGE) primitive camera routine, used by CAMPIX

SUM = INTOP(PIC, WINSIZE, ANSARRY, YEDGE(0), XEDGE(0)) interest operator

INTLOM(HIG, WID, ANSARRY) intop local max operator

SIZE = INTERESTDIM(PICTURE, WINDOWSIZE) pic size needed for interest op

INTEREST(PICTURE, WINDOW, RESULTPICTURE) make interest op picture

BESTVAL = MATCH(PICTURE1, SY1, SX1, SY2, SX2, PICTURE2, DY1, DX1, DY2, DX2) correlator, find Source window in pic1 in Dest in pic2

BSTCOEF = NORCOR(PICTURE1, SY1, SX1, SY2, SX2, PICTURE2, DY1, DX1, DY2, DX2) correlator, find Source window in pic1 in Dest in pic2

CLEAN(PICTURE) remove single pixel noise, blurs a little

PASSHI(PICTURE1, WINDOWSIZE, PICTURE2) high pass filter

LOWPAS(PICTURE) in place low pass filter. 4 pixels → 1 pixel.

SUM = CMPPAR(PICTURE1, PICTURE2) compare two pics $[\sum(x-y)^2]$

SUM = CMPPAD(PICTURE1, PICTURE2) quick and dirty compare

PERBIT(PICTURE, TRANSFORM) transform each pixel of pic

HISTOG(PICTURE, HISTOGRAM) count # of occurences of each gray val

ENHANCE(PICTURE) make histogram flat

SYNCHRONIZE(PICTURE1) do a vertical roll fixup

ROWSUM(PICTURE1, ROWSUMS) sum up the pixels in each row

ROWSUD(PICTURE1, ROWSUMS) dirty rowsums, one pixel/word used

COLSUM(PICTURE1, COLSUMS) sum up the pixels in each col

LONG REAL= SUMSQR(PIC) double prec. sum of squares of pixels

MASS = CENTRO(PIC, YL, XL, YH, XH, THR) centroid and moment of a dark area

UNPACK(SOURCEARRAY, PICTURE) copy a dense byte arry into a pic

GETPAR(ARRY, PICTURE) copy full word array of pixels to pic

PUTPAR(PICTURE, ARRY) copy pic to full word array of pixels

EDGEINIT(PICTURE, SIZE) initialize edge operator

EDGE(X, Y, EDGERESULT) apply edge operator

NOTE: all picture and other arrays are zero origin in all dimensions

Vision routines for Displays

VIXHDR.SAI for displaying gray scale and halftone pictures on data disc. an extension for the display routines in DDSUB.SAI[GRA,HPM]

requires: PIXHDR.SAI[VIS], DDHDR.SAI[GRA], VIXFAI.REL[VIS], VIXSAI.REL[VIS]

VIDEO(X1, Y1, X2, Y2, PICTURE, BIT) display PICTURE between X1, Y1, X2, Y2 in SCREEN co-ordinates. If BIT=-1 then a fast, low quality halftone, if -2 then a high quality halftone, if -3 then a buggy halftone. If positive then BIT represents a bit mask, which is anded with each pixel. If the result is nonzero, a bit is turned on in the corresponding part of the display.

VIDONE(PICTURE, BT, I(0), J(0)) similar to VID but faster and simpler. Maps picture elements one to one to data disc points. Upper left corner of picture is placed I physical DD scanlines from the top of the picture, and indented J DD elements from the left. Complements the masked bit instead of setting it to one.

VIDFOR(PICTURE, BUF1, BUF2, BUF4, BUF8, I(0), J(0)) Like VIDONE, but for 4 bit pictures and four DD buffers. Sets up all buffers at once, clearing displayed area and inverting bits for compatibility with the inverted gray code produced by TVRAW.

VIDFRT(PICTURE, BUF1, BUF2, BUF4, BUF8, I(0), J(0)) Transposed VIDFOR; picture twice as wide on its side.

VIDFRX(PICTURE, BUF1, BUF2, BUF4, BUF8, I(0), J(0)) Like VIDFOR, but makes picture twice as wide and tall. One picture pixel → 4 DD pixels.

VID1(PICTURE, BUF1, I(0), J(0)) Like VIDFOR, for 1 bit pictures, but assumes normal gray code and produces a complemented display.

VID3(PICTURE, BUF1, BUF2, BUF4, I(0), J(0)) Like VID1, for 3 bit pictures.

VID4(PICTURE, BUF1, BUF2, BUF4, BUF8, I(0), J(0)) Like VID1, for 4 bit pictures.

VID5(PICTURE, BUF1, BUF2, BUF4, BUF8, BUF16, I(0), J(0)) Like VID1, for 5 bit pictures.

SUCCESS = VIDXGP(PIC, I0, J0, PLEN) Send a picture to the XGP. Dumb

thing to do except for one bit pictures. Wait if XGP busy.

SUCCESS = VIDXG(PIC, I0, J0, PLEN) VIDXGP, but return with failure if XGP busy

SUCCESS = VIDXGQ(PIC, I0, J0, PLEN) VIDXGP, but set up detached job to do XGPing if XGP busy

Routines for putting Fonted Text into Pictures

FNTHDR.SAI for inserting XGP font characters into pictures.

requires: FNTFAI.REL[VIS], FNTSAI.REL[VIS]

FNTSEL(FNTNUM, FILSPEC, FNTHEAD) define font number FNTNUM to be font FILSPEC. FNTHEAD is the first word of an array '204 words long which must be reserved for this font.

CHRDEP(FNTNUM, CHR, PIC, YLO, XLO, YCOMP, XCOMP) add character CHR to the picture PIC in font # FNTNUM starting at position YLO, XLO compressed by YCOMP in Y and XCOMP in X.

CHRPED(FNTNUM, CHR, PIC, YLO, XLO, YCOMP, XCOMP) add CHR to PIC, sideways, writing bottom to top. X and Y positions and compressions refer to text, not picture, reference system.

CHR3X2(FNTNUM, CHR, PIC, YLO, XLO) like CHRDEP, but compresses X by 3 and Y by 2, and goes faster

CHR6X4(FNTNUM, CHR, PIC, YLO, XLO) like CHRDEP, but compresses X by 6 and Y by 4, and goes faster

CHR3Y4(FNTNUM, CHR, PIC, YLO, XLO) like CHRPED, but compresses X by 3 and Y by 4, and goes faster

FCACHE(BUFFER, BUFSIZ) set up a buffer for caching letter descriptions. Doing this greatly speeds up CHRDEP. 5 or 10 K is a good buffer size.

Defines FNTHIG, position in FNTHEAD where height is stored, and FNTBAS, where baseline is stored.

Internal Picture Array Format

WORD	CONTENTS	
0	PCLN	number of scanlines in the picture
1	PCWD	words in the pixel portion of the picture
2	PCBY	valid bytes in the picture
3	PCBYA	bytes in the picture, including null bytes at end of each scanline
4	LNWD	words per scanline
5	LNBY	valid bytes per scanline
6	LNBYA	bytes per scanline, including the nulls
7	WDBY	bytes per word
8	WDBI	bits in the valid portion of each word
9	BYBI	bits per byte
10	BMAX	$2^{BYBI} - 1$, the maximum value of a byte
11	BPTAB	address of SECOND entry in byte pointer table, 13+PCLN+address of array
12 to 11+PCLN	LINTAB	table containg the actual address of the first word of each scanline, in top to bottom order
12+PCLN to 12+PCLN+LNBYA		table containing byte pointers to samples within lines, to be added to line address. The first entry, when ILDB'ed causes loading of the first byte in the line.
13+PCLN+LNBYA to 12+PCLN+LNBYA +PCWD		the picture

Picture File Format

Simplified hand-eye file format, as written by PIXSAI routines, for a picture HIG samples high by WID samples wide by BIT bits/sample:

The first 200_8 word disk block of the file contains the following seven words of data (the rest of the block is unused).

WORD	CONTENTS	
0	-1	This identifies the file as a standard Stanford Hand Eye picture file
1	BIT	Number of bits/sample
2	$\{WID/[36/BIT]\}$	# of words/scanline. [] is FLOOR function, {} is CEILING operator
3	1	first scanline number
4	HIG	last scanline number
5	1	first column number
6	WID	last column number

The data begins on word 200_8 of the file, $\{WID/[36/BIT]\}$ words per scanline, left to right, top to bottom, for HIG scanlines. Each scanline begins on a word boundary.

The data compressed variant has the same header information except word 0 is -2 instead of -1. For each successive block of 36 words in a standard file a compressed file has from 1 to 37 words. Each bit of the first word in such a group represents one of the 36 words in the standard file block, sequenced left to right. The bit is zero if the corresponding word is the same as the previous word in the

file, or one if it differs. Each word that differs is given in the group that follows the mask word.

XGPSYN and XGPSYG

XGPSYN displays files on the video synthesizer, imitating the XGP. It can fill the screen with 1/2, 1 or 2 pages at time. XGPSYN can also list documents on the XGP, with no complexity limit, and make hand/eye compatible picture files which can be sent to printers like the VARIAN, or displayed on screens. XGPSYG has the added capability of inserting drawings and pictures into the assembled pages.

To run the programs, tell them which file you want to look at, either in the command line (R XGPSYN;FILENAME) or in answer to the FILE: question, and any spooler style switches, such as /FONT=BASL30 (.XGP files already contain most necessary switches). The page number questions can be answered with the page you want to view, any additional spooler switches, or one of the following commands:

XGPSYN	COMMANDS
H	Half density. Next display will be one half page per screen.
F	Full density. A whole page per screen.
D	Double density. Two sequential pages per screen.
C	Display pages on your own DD channel instead of on the video synthesizer.
S	Use the video synthesizer instead of your own channel.

V	view. Redraw the last display (in case it was clobbered).
W	negate subsequent displays. Black on white becomes white on black, and vice versa.
K	kill. Erase the video synthesizer.
Q	quit. Exit from the program and load the line editor with an XSPOOL command.
L(a:b)	list pages a to b on the XGP. More tolerant than XSPOOL, but slower. Alternate forms for this command are L to list the whole document and L(a) to list a single page.
B	Bitwise resolution. Next display will be a full size bit raster suitable for XGPing or sending to a picture file for printing on other devices.
T	Transposed bitwise resolution. Like B mode, but page is generated 90° rotated.
E	Enormous resolution. Next display w'll be at bit raster resolution, but only upper left 480 by 512 pixel portion will be generated.
O	output the last display as a data compressed picture file.
P	output the last display as a standard hand/eye picture file.
X	XGP output the last B,T or E display.

The video synthesizer is a video rate D/A driven by data disc channels 30 through 37. H density requires 3 of these channels, F needs 4 and D wants 5. These are rarely available during heavy system load. It takes about 6 CPU seconds to compose a single page.

XGPSYN and XGPSYG understand the following switches, some of which are not standard with the spooler or the COPY program. The switch names may be abbreviated to the capitalized portion.

/FONT="fontname" Select font number 0 for the document

/FONT#n="fontname" Select font number n for the document

/THickness=t Select line thickness for Leland's music files

/ESCape=... change the escape sequence. Any characters except slash are ok.

/REpeat=n When listing, make n copies of each page.

/TMar=n Set the top margin of a listing n raster lines from top of page.

/PMar=n Set the text portion of the page to be n lines tall

/BMar=n Make the bottom margin n raster lines big. In listings the sum of TMAR+PMAR+BMAR is the physical length of the page.

/LMar=n Set the left margin n pixels from left edge of paper.

/RMar=n Set the right margin n pixels from the left edge of the paper. When writing picture files of page images, RMAR is the physical width of the resulting image.

/List List the document on the XGP. Possible forms are /L to list the whole document, /L(n) to list page n and /L(a:b) to list all the pages between a and b. The simplest way to list a document with XGPSYN is by incanting R XGPSYN;file/L

/XLine=n Insert n extra scanlines between lines of text. This number is initially 3.

/INterchar=n Insert n extra columns between characters of text. This number is usually zero.

/XShift=n Shift the contents of a page n pixels to the right on the image. Useful if you want to tweak the margins in a listing, and also for making images too large to fit in XGPSYN's core image all at once. Set RMAR small, the output the same page repeatedly with different XSHIFTS. Resulting windows can be combined later into a single file representing a large page.

/YShift=n Shift the page contents n pixels up. For tweaking vertical margins, and also for making very long pages. Set TMAR+PMAR+BMAR small, then window through the file by changing YSHIFT. If resulting windows are to be assembled later the EDGE switch is also recommended.

/EDge Normally characters that extend past the top margin of a page are not displayed. /EDGE selects a slower mode in which such fractional charac-

ters do appear. This is necessary if large pages are to be assembled from small windows.

/XGp This file is in XGP format, whether or not the file extension says so.

/NOXgp This file is not in XGP form (is not preceded by a switch page)

/NOQueue When listing on the XGP, XGPSYN will create a detached job which waits if the XGP is not available when a page is to be generated. NOQUEUE supresses this feature. Instead, XGPSYN itself waits for the XGP.

/NODpy Supresses v. leo synthesizer display. Useful if XGPSYN is being used solely to generate files containing page images. /L invokes this mode automatically.

/AUTocr Insert carriage returns when lines run beyond right margin.

/NOAutocr Suppress insertion of extra carriage returns.

/Halfdensity Select half page/screen mode.

/Fulldensity Select full page/screen mode.

/Doubledensity Select two page/screen mode.

/Enormousresolution In this mode a screenful of display is generated without any compression of the original page raster. Very little of a standard page is visible, but every pixel can be resolved in that portion.

/Bitwisedensity Create a one bit/pixel image of the whole page. This can be sent to the XGP (/L uses this density) or written into a hand-eye picture file. Such files can be listed on other printing devices.

/Transposedbitwisedensity Like /B, but the image comes out on its side, rotated 90°.

/Varian Useful only with XGPSYG. Causes halftones to be generated in a high density mode which works well with the Varian printer, but causes washing out on the XGP

GOD Files and XGPSYG

The following program writes the GOD file that produced Figure 10-1 in Chapter 10.

```
BEGIN "PRETTY"
REQUIRE "TYPHDR.SAI[GOD,HPW]" SOURCE_FILE;

INTEGER FJ,I,J,K,L,M,N; REAL P,Q;
REAL ARRAY X,Y[1:10];

FJ←FILJOB("DSK:PRETTY.GOD[DIA,HPW]");
                                        comment open the GOD file;
DDINIT; SCREEN(-1.2,-1.2,1.2,1.2);

PICFIL(-1,-1,1,1,"U:SF2.PIC[DIA,HPW]"); LITEN;
                                        comment insert a picture;
LINE(-1,-1,-1,1); LINE(1,1,-1,1);
LINE(1,1,1,-1);   LINE(-1,-1,1,-1);
                                        comment outline it;
FNTSELECT(2,"METMBW"); FNTSELECT(3,"METSBW");
FNTSELECT(103,"BASL30");
                                        comment select some fonts;
FNTPOS(-1,1.03,1,1,0,0);
FNTEXT(0,0,2,"See the pretty aeroplane");
                                        comment use font 2;
DRKEN;
   BEGIN REAL ARRAY X,Y[1:20]; INTEGER I;
   FOR I←1 STEP 1 UNTIL 20 DO
     BEGIN X[I]←.89+0.304*COS((I-1)*2*3.14159/20);
     Y[I]←.5+0.12*SIN((I-1)*2*3.14159/20); END;
   X[12]←.62; Y[12]←-.23;
   POLYGO(20,X[1],Y[1]);
   LITEN;
   FOR I←1 STEP 1 UNTIL 20 DO
      LINE(X[I],Y[I],X[(I MOD 20)+1],Y[(I MOD 20)+1],3);
   END;
                                        comment make a balloon;
FNTPOS(.89,.5);
DEPOSIT(0,0,CENTER(JTXT(3,"Yow !!!  I am an L1011 !!")));
                                        comment put font 3 text into it;
DPYUP(-1); KILJOB(FJ);                  comment close the GOD file;
END;
```

The GOD file contains graphics commands like line, dot, text, picture

etc. When PRETTY.SAI says LINE(...) a line command gets written into PRETTY.GOD. When it says PICFIL(..., "SF1.PIC") a binary rendition of the command gets written into the file. It is the job of whatever program reads PRETTY.GOD to deposit the picture when it encounters the PICFIL command in it, just like it's its responsibility to draw a line when it sees a LINE command.

You can insert a .GOD file into as document with XGPSYG by including a line of the form ⊂⊗⊃G[0,.5](5,4):PRETTY.GOD[GOD,HPM]⊂⊗⊃ in your text.

This means	
⊂⊗⊃	Here is an escape
G	its a GOD file escape
[0,.5]	diagram center is 0 in right and .5 above page center
(5,4)	diagram is to be 5 inches wide by 4 inches high
PRETTY.GOD	get your graphics commands from this file
⊂⊗⊃	Here is the end of the escape

The position field in square brackets and the size field in parens are optional. If left out, the picture will be centered around where your escape text would have appeared if you had XSPOOLED'd or XGPSYN'ed your document. An alternative form for the position field is [%-2.3,%+3.7], which means the center of the diagram is to be put 2 3 inches to the left and 3.7 inches above where your escape sequence would have been deposited. Thus you can position diagrams on the page either absolutely, or relative to the position of the escape sequence. It is ok to make the X position, say, relative and the Y position absolute.

Connections

Philosophy

Artificial Intelligence researchers are like the blind men who went to see the elephant. Having of necessity experienced only tiny portions of the situation, each comes to a different conclusion about the nature of the whole.

One group suggests that the construction of an intelligent machine is very like mathematics, and finding the "theorems" of intelligence will involve clever representations, transformation rules and long lemmas, gotten at mostly by thinking hard.

Another school feels AI is like theoretical physics, and the solution involves finding the universal "laws of intelligence", by means of theories guided by experiment.

Yet another sees AI as a little like biology, the idea being to explain intrinsically complicated natural mechanisms as simply as possible.

A fourth treats AI as a problem in psychological introspection, transferring rules of conscious thinking into mechanical form.

A fifth feels the problem is one of engineering, with an artificial intelligence being just another big machine, to be built subsystem by subsystem, by rule of thumb and experience based intuition.

People's points of view change with experience and mood, and most of us have found ourselves espousing different approaches at different times.

The AI effort has a specific and lofty goal, the matching of human performance in intellectual and other tasks by machines. Because the overall goal is still far from accomplished, many of us suffer from doubts about our progress. These often expresses themselves in the feeling that much of our field is somehow not "scientific". Depending on our mood, this transforms to "not like mathematics", or "not like physics" etc.. And of course its easy to find many projects that fail to meet our arbitrary standards, and confirm our suspicions.

The hard sciences are distinguished from many other intellectual pursuits not by the quality of the workers, or even the methods employed, but by the amount of independent verification and refutation practiced. It is the ruthlessness of the evaluation function that separates the useless from the valuable and the capable from the incompetent.

I feel it is too early to commit ourselves to or to excessively condemn any of the various approaches. We ought to judge AI programs on the basis of performance. Whether or not they conform to our theory of the moment as to what constitutes intelligence and how to go about building it, or what is esthetic, we should ask "how well does it work?".

In other words, I think AI is very like evolution. We should try different modifications and approaches and see which ones prove themselves experimentally.

Since this is in itself a prejudgement, I don't really want to force it on others. But if we suspend disbelief for a few minutes, I can use it to show why roving vehicles are on the direct path to human equivalence. The argument is by analogy with natural evolution.

Locomotion, Vision and Intelligence

Consider that, with few exceptions, the only natural systems with AIish capabilities are large mobile animals. An apparent minimum size for nerve cells explains the complexity limits on small animals like insects. The role of mobility in the development of imaging vision and intelligence is more subtle, yet real. No plants or sessile animals (what few there are) have imaging eyes or complex nervous systems, but there are several independent instances of vision and comparative intelligence in the presence of mobility.

The evolutionary mainstream (as defined by us mainstreamers), fishes through amphibians and reptiles to mammals to us, is one such instance. Imaging eyes and a moderate brain developed roughly simultaneously with a backbone, in motile protofish, sometime in the Paleozoic, about 450 million years ago. Brain size changed little through the slow moving amphibian and reptile stages, then accelerated sharply with the transition to the more mobile mammalian form, about 100 million years ago.

Instance two is the birds, who also have reptilian ancestry, and who's development parallels our own. Though size limited by the dynamics of flying, several bird species can match the intellectual performance of all but the smartest

mammals. The battle of wits between farmers and crows is legendary, and well documented. The intuitive number sense of these birds goes to seven, compared to three or four with us (without counting). Hard evidence comes from "reversal learning" experiments The response giving the reward in a Skinner box is occasionally inverted. Most animals are confused by the switch, and actually take longer than the first time to learn the new state (as if they first had to unlearn the old rules). Primates (monkeys and apes and us) among mammals, and virtually all birds, on the other hand, "catch on" after the first reversal, and react correctly almost instantly on later swaps.

Instance three is surprising. Most molluscs are nearly blind, intellectually unimpressive, very slow moving shellfish. Their relatives who opted for mobility, the cephalopods (octopus and squid) provide a dramatic contrast, having speed, good eyes, a large brain, a color display skin, mammal-like behavior, and even manipulators. The similarities to mammals are especially significant because they were independently evolved. Our last shared ancestor was a billion year old bilaterally symmetric pre-worm, with a few neurons. The differences are interesting The eyes are hemispherical and firmly attached to the surrounding skin, and the light sensitive cells in the retina point outwards, towards the lens. The brain is annular, encircling the esophagus, and is organized into several connected clumps of ganglia, one for each arm. A Cousteau film documents an octopus' response to a "monkey and bananas" problem. A fishbowl sealed with a large cork, and containing a small lobster, is dropped into the water near the animal. The octopus is immediately attracted, seemingly recognizing the food by sight. It spends a while probing the container and attempting to reach the lobster from various angles, unsuccessfully Then, apparently purposefully, it wraps three or four tentacles around the bowl, and one about the cork, and pulls. The cork comes free and shoots to the surface, and the octopus reaches a free tentacle into the bowl to retrieve the lobster, and eats.

The Point

The point of the preceeding ramble is; moving through the wide world is a demanding task, and encourages development of complex responses in those who undertake it. Moving organisms (and machines) must learn to deal with a wide variety of situations, and have many responses open to them. This variety places a great premium on general techniques, and makes highly specialized methods, which may be optimal for sessile creatures, less valuable. These forces seem to have led to relative intelligence in animals. Perhaps they mark one route to the same goal for machines.

References

[A1] **Arnold, R.D.,** *Local Context in Matching Edges for Stereo Vision,* Proceedings: Image Understanding Workshop, (L. S. Baumann, ed.), Science Applications, Inc., Cambridge, Mass., May 1978, 65-72.

[B1] **Brooks, Rodney A.,** personal communication, Stanford University, 1977.

[B2] **Binford, Thomas O.,** personal communication, Stanford University, 1975.

[B3] **Brooks, Rodney A.,** *An Onboard Microprocessor for The Cart,* unpublished, Stanford University, December 1977.

[B4] Baker, H. Harlyn, *Three-Dimensional Modelling,* Proceedings of the fifth IJCAI, MIT, Cambridge, Mass., August 1977, 649-655.

[B5] **Baumgart, Bruce G.,** *Geometric Modelling for Computer Vision,* (Ph.D. thesis), Stanford AI lab memo AIM-249, October 1974.

[BC1] **Burr, D.J., and R.T. Chien,** *A System for Stereo Computer Vision with Geometric Models,* Fifth International Joint Conference on Artificial Intelligence, MIT, August 1977, 583.

[D1] **Dijkstra, E.W.,** *A Note on Two Problems in Connection with Graphs,* Numerische Mathematik, 1, 269-271, 1959.

[G1] **Gennery, D.B.,** *Object Detection and Measurement Using Stereo Vision,* Sixth International Joint Conference on Artificial Intelligence, Tokyo, August, 1979, 320-327.

[G2] **Gennery, D.B.,** *A Stereo Vision System for an Autonomous Vehicle,* Proceedings of the 5th IJCAI, MIT, Cambridge, Mass., 1977, 576-580.

[GM1] **Grimson, W.E.L. and D. Marr,** *A computer implementation of a theory of human stereo vision,* Proceedings of the April 1979 Image Understanding Workshop, 41-47.

[H1] **Hannah, M. J.,** *Computer Matching of Areas in Stereo Images,* Stanford Artificial Intelligence Laboratory, AIM-239, July 1974.

[HNR1] **Hart, P., N.J. Nilsson, and B. Raphael,** *A formal basis for the heuristic determination of minimum cost paths,* IEEE Transactions on Systems Science and Cybernetics, SSC-4, 2, July 1968, 100-107.

[J1] **Julesz, B.,** *Foundations of Cyclopean Perception,* University of Chicago Press, Chicago, 1971.

[K1] **Knuth, Donald E.,** *Tau Epsilon Chi, A System for Technical Text,* Stanford AI lab memo AIM-317, September 1978.

[L1] **Lowry, Michael,** personal communication, Stanford University, 1980.

[LOY1] **Levine, M.D., D.A. O'Handley, and G.M. Yagi,** *Computer Determination of Depth Maps,* Computer Graphics and Image Processing 2, 1973, 131-150.

[LW1] **Lozano-Pérez, Tomás and Michael A. Wesley,** *An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles,* CACM 22, number 10, October 1979, 560-570.

[M1] *Macsyma Reference Manual,* version 9, The Mathlab Group, Laboratory for Computer Science, MIT, Cambridge, Mass. December 1977.

[M2] **Moravec, H.P.,** *Visual Mapping by a Robot Rover,* Proceedings of the 6th International Joint Conference on Artificial Intelligence, Tokyo, Japan, August 1979, 589-601.

[M3] **Moravec, H.P.,** *Towards Automatic Visual Obstacle Avoidance,* Proceedings of the 5th IJCAI, MIT, Cambridge, Mass., 1977, p. 584.

[MP1] **Marr, D. and T. Poggio,** *Cooperative Computation of Stereo Disparity,* Science 194, Oct 1976, 283-287.

[MP2] **Marr, D. and T. Poggio,** *A Computational Theory of Human Stereo Vision,* MIT AI lab memo 451.

[MKA1] **Mori, K., M. Kidode, and H. Asada,** *An Iterative Prediction and Correction Method for Automatic Stereocomparison,* Computer Graphics and

Image Processing 2, 1973, 393-401.

[N1] **Nevatia, R.K.,** *Depth Measurement by Motion Stereo,* Computer Graphics and Image Processing, 5, 1976, 203-214.

[N2] **Nitzan, D.,** *Stereopsis Error Analysis,* Artificial Intelligence Center Technical Note 71, SRI International, Menlo Park, Ca., September 1972.

[PT1] **Pingle, K.K., and A.J. Thomas,** *A Fast, Feature-Driven Stereo Depth Program,* Stanford Artificial Intelligence Laboratory, AIM-248, May 1975.

[Q1] **Quam, L.H.,** *Computer Comparison of Pictures,* Stanford Artificial Intelligence Laboratory, AIM-144, May 1971.

[QH1] **Quam, L.H., and M.J. Hannah,** *Stanford Automatic Photogrammetry Research,* Stanford Artificial Intelligence Laboratory, AIM-254, December 1974.

[R1] **Reiser, John F.,** *SAIL,* Stanford AI lab memo AIM-289, August 1976.

[S1] *Microbroadcasting,* Scientific American, Vol. 242 #5, May 1980, pp 84-87.

[S2] **Schmidt, R.A.,** *A Study of the Real-Time Control of a Computer Driven Vehicle,* (PhD thesis), Stanford AI Memo AIM-149, August 1971.

[W1] **Westphal, Harald,** personal communication, Stanford university March 1979.

[W2] **Weyrauch, Richard W.,** personal communication, Stanford University, 1975

[WG1] **Wright, F.W. and R.E. Gorin,** *FAIL,* Stanford AI lab memo AIM-291, October 1976

[YC1] **Yakimovsky, Y. and R. Cunningham,** *A System for Extracting Three-Dimensional Measurements from a Stereo Pair of TV Cameras,* Computer Graphics and Image Processing 7, 1978, 195-210.